Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação da ferramenta de testes Selenium no desenvolvimento guiado por teste de uma aplicação web

Rafael Porto Santori

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Fernando Albuquerque

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação da ferramenta de testes Selenium no desenvolvimento guiado por teste de uma aplicação web

Rafael Porto Santori

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Fernando Albuquerque (Orientador)
CIC/UnB

Prof. ª Dr. ª Fernanda Lima
CIC/UnB

Prof. ª Dr. ª Genaina Nunes Rodrigues
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 02 de dezembro de 2019

Dedicatória

Dedico este trabalho a todas as pessoas que contribuíram para a minha formação acadêmica, direta ou indiretamente. Aos professores e aos colegas, atores importantes para construção conjunta do conhecimento, sendo muitas vezes facilitadores nesse processo de crescimento e aperfeiçoamento pessoal. Em especial, aos meus pais e à minha namorada, por todo o apoio, paciência e carinho durante todo o longo e cansativo período de estudos.

Agradecimentos

Agradeço à minha família por ter me dado condições de ter uma educação de qualidade, aos meus colegas de trabalho pela parceria no aprendizado diário e aos professores da universidade, especialmente ao professor Fernando Albuquerque, por ter aceitado ser o orientador deste trabalho e por todo o apoio durante o seu processo de desenvolvimento.

Resumo

O teste de software é um processo importante no desenvolvimento de software. Em algumas situações, a automação desse processo pode representar benefícios importantes para o controle de qualidade do software. Nesse contexto, existem diversas ferramentas de automação de testes que buscam facilitar a atividade de automação. O presente trabalho procura desenvolver um arcabouço para avaliação do Selenium, uma ferramenta de automação de testes de aplicações web. Para atingir esse objetivo, inicialmente alguns critérios de avaliação de ferramentas foram definidos para que, após o desenvolvimento de uma aplicação web por meio do processo de desenvolvimento guiado por testes (TDD), o uso da ferramenta de automação de testes fosse então avaliado.

Palavras-chave: TDD, aplicação web, teste, avaliação, Selenium

Abstract

Software testing is an important process of software development. In some situations, automating this process may produce some important benefits for software quality control. Therefore, there are several test automation tools that aims to facilitate automation of software testing. This work aims to evaluate Selenium, a test automation tool for web application. To achieve this goal, some evaluation criteria was defined to evaluate the automation testing tool after the development of a web application in which the Test Driven Development (TDD) process was used.

Keywords: TDD, Web Application, Test, Evaluation, Selenium

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O teste de software é um importante processo no controle da qualidade de software. Uma das responsabilidades desse processo é a verificação dos componentes, ferramentas e atributos de qualidade de um software, avaliando a sua conformidade com requisitos funcionais e não funcionais do software. Esse processo é realizado a partir de determinadas entradas (*inputs*) de dados, de modo que os devidos processamentos sejam executados e gerem os resultados necessários para verificação do software. Apesar dessa dinâmica ser de simples caracterização, o processo de teste pode representar um alto nível de complexidade, gerando custos elevados e demandando um grande período de tempo para realização [10].

Devido à importância e complexidade dos testes de software, é necessário evoluir esse processo para que ele seja realizado de maneira mais rápida. Uma alternativa para que os testes sejam realizados de maneira mais rápida é a utilização de ferramentas de automação de testes, reduzindo o envolvimento dos desenvolvedores e analistas de testes nesse processo [10]. A automação dos testes ganha ainda mais relevância em algumas técnicas de desenvolvimento, como o processo de desenvolvimento guiado por testes (TDD), onde o desenvolvimento de uma funcionalidade do software é iniciado pela geração dos seus casos de teste e a sua conclusão está condicionada à aceitação do teste [11].

Para facilitar a automação de testes, existem diversas ferramentas cujo propósito é a execução iterativa dos testes do software de forma automática. Muitas delas possuem recursos interessantes para geração de dados de entrada variados e exibição dos resultados. Entretanto, cada ferramenta pode ser destinada ao trabalho em determinadas fases do processo de teste, e adotar linguagens de programação e tecnologias específicas. Portanto, é relevante verificar as características da ferramenta de automação de testes com o intuito de avaliar a sua aderência ao contexto em que ela será utilizada e se não existe alguma alternativa melhor a ser utilizada [7].

Selenium pode ser caracterizado como uma ferramenta para automação de testes de

aplicações web [12]. Além de ser portável, com integração disponível para diversos navegadores web, sistemas operacionais e linguagens de programação, o Selenium é um software de código aberto, garantindo a sua acessibilidade a todos que queiram realizar um projeto de automação de testes. Os seus múltiplos recursos fazem com que o Selenium seja uma ferramenta interessante e consolidada para realização de testes de software.

1.1 Problema

Durante o ciclo de desenvolvimento de software, um relevante processo é o de teste, responsável por verificar se o software desenvolvido está de acordo com os seus requisitos. Apesar de ser importante para controlar a qualidade do software, o processo de teste de software pode levar muito tempo para ser concluído. Além disso, esse processo possui diversas naturezas e propósitos distintos.

Para facilitar a realização dos testes, algumas ferramentas específicas foram desenvolvidas com o intuito de realizar a automação dos testes, possibilitando a execução do teste de software de maneira mais rápida. Muitas dessas ferramentas de automação possibilitam a configuração de cenários específicos para o teste e geração de relatórios de resultado. Por isso, é relevante avaliar a ferramenta de automação de testes com base no contexto em que ela será utilizada para assegurar que os testes sejam realizados da forma adequada.

1.2 Objetivo geral

Avaliação das vantagens e desvantagens do uso da ferramenta de testes Selenium na automação de testes funcionais durante o desenvolvimento de parte de uma aplicação web utilizando processo de desenvolvimento guiado por testes (TDD).

1.3 Objetivos específicos

Os objetivos específicos do projeto são:

- Descrever conceitos sobre aplicações web;
- Descrever conceitos sobre o processo de desenvolvimento guiado por testes (TDD);
- Desenvolver um arcabouço para avaliação da ferramenta de testes Selenium com base em critérios de avaliação de ferramentas de automação de teste.

1.4 Justificativa

O teste de software é um processo essencial para controlar a qualidade do software desenvolvido e a sua aderência aos requisitos funcionais e não-funcionais. Considerando que os testes podem apresentar problemas de execução no que se refere ao tempo necessário para a sua realização, a automação dos testes surge como uma alternativa interessante para facilitar essa atividade e garantir a entrega do software com qualidade e de maneira mais rápida. Já que existem diversas ferramentas disponíveis para a automação de testes, cada uma com características e propósitos específicos, é importante avaliar as ferramentas para verificar os seus pontos positivos e negativos.

A aplicação web é uma classe de software que suporta o compartilhamento de diversas mídias diferentes e a integração com várias ferramentas. Sendo assim, devido à tecnologia suportada por essas aplicações e as funcionalidades já disponibilizadas para elas, a aplicação dos testes para essa classe de software é interessante, aliado ao fato de que essas aplicações tipicamente passam por constantes transformações durante o ciclo de vida das mesmas.

Por fim, o processo de desenvolvimento guiado por testes (TDD) permite combinar a realização de testes durante o desenvolvimento do software. Nesse contexto, a automação dos testes é importante, justificando assim a avaliação da ferramenta de automação de testes durante o desenvolvimento de uma aplicação web utilizando o TDD.

1.5 Estrutura do projeto

O projeto está dividido em fundamentação teórica, abordagem prática e conclusão do trabalho. Como esse é um projeto de licenciatura, buscou-se a elaboração de um referencial teórico mais completo e informativo, podendo ser utilizado como referência para o aprendizado dos conteúdos.

A fundamentação teórica é composta por cinco capítulos, sendo baseada em diversas fontes de informação. O capítulo 2 apresenta conceitos, características e tecnologias relacionadas com as aplicações web. No capítulo 3 são apresentados conceitos sobre o processo de desenvolvimento guiado por testes, abordando um histórico sobre o assunto e a descrição dos principais modelos de processo de software guiados por testes. O capítulo 4 apresenta informações sobre o processo de automação de testes no contexto do desenvolvimento guiados por testes, descrevendo algumas ferramentas que podem ser utilizadas com essa finalidade. O capítulo 5 aborda os critérios que podem ser utilizados para avaliação de ferramentas, especialmente para as ferramentas de automação de testes. O capítulo

6 apresenta as principais características do Selenium, uma ferramenta de automação de testes.

A abordagem prática do trabalho é composta por três capítulos. O capítulo 7 apresenta as informações referentes ao planejamento do projeto, descrevendo o contexto em que os testes foram realizados e os critérios de avaliação da ferramenta de automação definidos a partir da literatura. Já no capítulo 8 são descritos detalhes da implementação e execução dos testes, além das ferramentas utilizadas durante o projeto. Após a abordagem prática, no capítulo 9 encontra-se uma análise sobre a ferramenta de teste utilizada para a automação dos testes durante o projeto, baseada nos critérios de avaliação definidos previamente. Por fim, são apresentadas as últimas considerações acerca do trabalho desenvolvido e também são sugeridos possíveis trabalhos futuros.

Parte I

Referencial Teórico

Capítulo 2

Aplicações Web

Nesse capítulo, que possui o objetivo de abordar as aplicações web, são apresentados alguns conceitos, características e tecnologias relacionadas a essa classe de software. Desta forma, inicialmente o conceito de aplicação web é descrito, relacionando-o com o histórico do seu desenvolvimento. Posteriormente, algumas informações da discplina Engenharia Web são elencadas, buscando caracterizar os principais aspectos relacionados ao desenvolvimento de sistemas baseados na web, dando enfoque ao processo de teste de aplicações web.

2.1 Definição e histórico

A web é o termo comum utilizado para caracterizar a *World Wide Web* [13]. A sua origem está baseada em uma demanda dos cientistas para compartilhar informações e documentos com cientistas de outras partes do mundo, uma vez que eles geravam dados de pesquisa em larga escala. Sendo assim, nas décadas de 1980 e 1990, essa rede de comunicação global foi proposta na Suíça como um repositório de informações que poderiam ser acessadas por diversos usuários [14].

Para possibilitar a transferência de arquivos entre diferentes usuários, o *Hypertext Transfer Protocol* (HTTP) foi criado, permitindo que um computador fizesse a requisição de dados de outro computador [14]. Com isso, os dados seriam disponibilizados para o usuário no computador que está fazendo a requisição. Esse funcionamento está baseado em um modelo cliente-servidor, onde o computador que requisita os dados é o cliente e o que possui os dados, disponibilizando-os após a requisição, é o servidor. Um mesmo computador pode atuar como servidor ou como cliente em momentos diferentes, dependendo do seu papel na troca de dados.

A linguagem utilizada para escrever os documentos na web, que contém dados e links para outros documentos, foi a *Hypertext Markup Language* (HTML). O seu conceito tam-

bém foi introduzido no contexto de criação da web, idealizada como uma linguagem que indicaria para os navegadores o conteúdo que deveria ser apresentado, sendo que ela continua apresentando relevância até hoje. Além do conteúdo a ser exibido, os documentos HTML também contém instruções de formatação, que permanecem sendo evoluídas constantemente para melhor comunicação, padronização e exibição pelos navegadores [14].

Outro componente importante que foi criado para a web foi o *Uniform Resource Locator* (URL), que é um sistema de nomeação para se referir aos documentos. O seu uso está baseado na nomeação de um objeto, tornando possível a sua identificação e localização no computador. O nome da URL é composto pela identificação do computador com o endereço IP, o documento no sistema de arquivos e um protocolo de comunicação [14].

Inicialmente, os documentos disponibilizados na web consistiam de simples arquivos de texto, com *links* estáticos para conexão com outras páginas. Entretanto, os engenheiros de software perceberam um grande potencial nessa estrutura, observando que ela poderia prover uma plataforma para aplicações com a execução ocorrendo no servidor [14]. Esse uso representa uma maior complexidade, já que os documentos não apresentariam mais um conteúdo fixo, e sim uma página web gerada dinamicamente com base no resultado de um processamento ocorrido no servidor. Nesse caso, o servidor poderia, por exemplo, retornar dados específicos de um banco de dados a partir das entradas (*inputs*) fornecidas pelo usuário, formatando-os em um documento HTML antes de enviar para o cliente.

As aplicações web podem então ser caracterizadas como sistemas projetados para serem utilizados com um navegador, utilizando o modelo cliente-servidor e tecnologias da web, como a linguagem HTML e o protocolo HTTP, para transferir documentos dinâmicos para o usuário, gerados como resultado de um processamento ocorrido no servidor [14]. Essas características garantem uma flexibilidade e reutilização de componentes, apesar de dificultarem o desenvolvimento das páginas web.

Com o passar do tempo e a consolidação desses recursos, a web se tornou uma plataforma bastante atrativa, seja para fins sociais ou comerciais. Nesse contexto, ela começou a ser usada como uma interface universal para as aplicações de negócios, sistemas informacionais e bancos de dados, por exemplo, apresentando um crescimento exponencial no seu uso [13]. Atualmente, ela suporta compartilhamento de diversas mídias, como foto, aúdio e vídeo, interações com o usuário, trabalho cooperativo, fluxos de trabalho, entre outras características mais complexas.

O crescimento da web está pautado em diferentes perspectivas e dimensões, tais como: o aumento da quantidade de páginas web, o aumento da quantidade de usuários, o aumento na quantidade de visitas, a interatividade e as funcionalidades disponibilizadas pelas aplicações web, as tecnologias utilizadas pelas aplicações web e o impacto social e comercial da web [13].

Portanto, a web possui muitas características particulares que devem ser consideradas para que o desenvolvimento de aplicações web seja feito de forma adequada. Uma característica que deve ser levada em consideração é que muitas aplicações web são evolucionárias e, por isso, necessitam de mudanças constantes no seu conteúdo. Outro aspecto relevante é que, como os usuários de aplicações web geralmente são um público diverso e global, o conteúdo deve ser de simples navegação, suportar diferentes formatos e arquiteturas, possuir diferentes linguagens e ainda ser atrativo [13]. Tudo isso deve ser feito sem negligenciar aspectos de segurança e performance.

Além disso, o processo de desenvolvimento deve considerar diversas etapas diferentes, sendo elas: planejamento, seleção de uma arquitetura web adequada, desenho do sistema, desenho da página, codificação, criação e manutenção de conteúdo, teste e avaliação de performance. É importante considerar que o sistema precisa de manutenção e atualização contínua, na medida que os seus requisitos são alterados [13].

2.2 Engenharia Web

A Engenharia Web é uma disciplina criada para trabalhar com os processos de desenvolvimento de aplicações e sistemas baseados na web, gerenciando a sua complexidade e diversidade para evitar possíveis falhas [2]. Nesse contexto, essa disciplina busca aplicar os princípios científicos, da engenharia e de gestão em abordagens sistemáticas para o desenvolvimento, implantação e manutenção das aplicações web de forma satisfatória [13].

Considerando as especifidades de uma aplicação web, a função da Engenharia Web é auxiliar na criação de uma infraestrutura que possibilite a evolução e manutenção do sistema. Sob essa perspectiva, ela precisa garantir uma visão holística e proativa para possibilitar o apoio no desenvolvimento de sistemas complexos, de modo que os desenvolvedores compreendam aspectos gerais e completos sobre o uso do sistema [13].

A abrangência da Engenharia Web está relacionada com uma série de processos, sendo eles: análise de requisitos; modelagem do sistema; arquitetura web; desenho do sistema; desenho da página web; codificação; interface com bancos de dados, sistemas *Enterprise Resource Planning* (ERP) e outras sistemas web; qualidade; usabilidade; segurança; avaliação de performance; teste; metodologias de desenvolvimento; processos de desenvolvimento; métricas; e gerenciamento de projetos [13]. Tudo isso está baseado no fato de que o desenvolvimento de aplicações web é um processo contínuo, começando na concepção, desenvolvimento, implementação e desempenho até a manutenção constante do sistema.

Com base na natureza das aplicações web, a Engenharia Web precisa de características multidisciplinares, envolvendo fatores de diversas áreas do conhecimento, que englobam: a

interação humano-computador, interface do usuário, análise e desenho de sistemas, engenharia de software, engenharia de requisitos, hipertexto, teste, gerenciamento de projetos, entre outros [1]. A Figura 2.1 representa esse aspecto multidisplinar da Engenharia Web, agregando elementos de outras áreas.

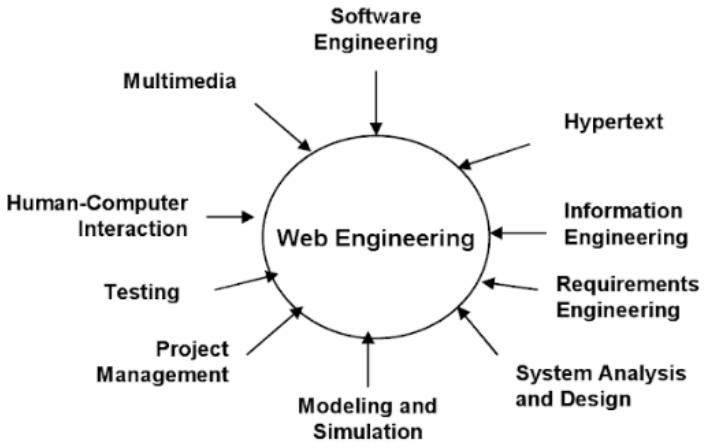

Figura 2.1: Multidisciplinaridade da Engenharia Web (Fonte: [1]).

O processo de desenvolvimento web deve, portanto, ser decomposto em diferentes fases, passos e atividades distintos, procurando minimizar os riscos, gerenciar as mudanças, atender os prazos e garantir o *feedback* do projeto. A sua efetividade também depende de um bom planejamento para que os desenvolvedores possam seguir etapas de uma maneira mais clara. Ginige e Murugesan [2] propõem um conjunto de 10 passos dependentes e iterativos para o sucesso no desenvolvimento e implantação de aplicações web, conforme lista a seguir.

1. Entender o funcionamento geral do sistema e o seu ambiente operacional, incluindo os objetivos de negócio e os requisitos;
2. Identificar claramente os *stakeholders*, que serão os principais usuários do sistema, a organização que precisa do sistema e o patrocinador do desenvolvimento;
3. Especificar os requisitos funcionais, técnicos e não técnicos dos *stakeholders* e do sistema. Importante destacar que esses requisitos tendem a ser alterados durante o desenvolvimento do sistema e a cada iteração ou revisão dos processos;
4. Desenvolver a arquitetura geral do sistema baseado na web, atendendo os requisitos técnicos e não técnicos;

5. Identificar subprojetos ou subprocessos para implementar a arquitetura. Caso possuam grande complexidade, dividi-los até que se tornem um conjunto de tarefas gerenciáveis;
6. Desenvolver e implementar os subprojetos;
7. Incorporar mecanismos efetivos de gerenciamento da evolução, mudança e manutenção do sistema web. À medida que o sistema for evoluindo, os passos anteriores devem ser repetidos, quando necessário;
8. Verificar os problemas não técnicos, como políticas organizacionais e de gestão, aspectos sociais, culturais e sociais, por exemplo;
9. Avaliar a performance do sistema;
10. Refinar e atualizar o sistema.

Ginige e Murugesan [2] também recomendam o uso de um processo de desenvolvimento web conforme explicitado na Figura 2.2, que auxiliaria na captura dos requisitos, permitiria a integração de diferentes áreas do conhecimento, facilitaria a comunicação entre os participantes do processo de desenvolvimento, suportaria a evolução e manutenção contínua e auxiliaria na gestão da complexidade do processo de desenvolvimento.

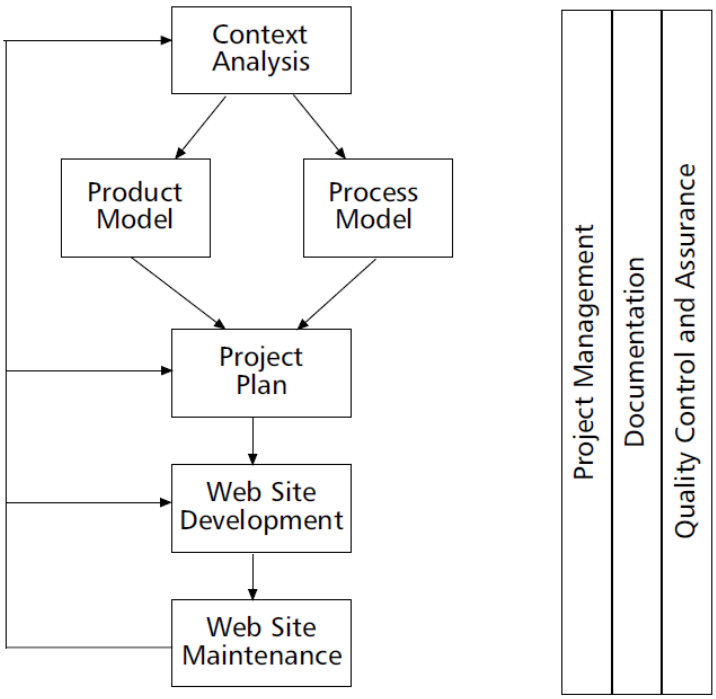

Figura 2.2: Processo de desenvolvimento para sistemas web (Fonte: [2]).

A primeira etapa do processo de desenvolvimento web proposto é a análise de contexto, atividade em que devem ser compreendidos os principais objetivos e requisitos do sistema, reunidas as informações sobre o ambiente operacional e identificados os *stakeholders* [2]. A execução desta etapa de maneira satisfatória pode evitar problemas de entrega de sistemas que não atendem os requisitos e não possuem as funcionalidades necessárias.

A segunda etapa é representada por uma bifurcação, onde temos o modelo do produto e o modelo do processo. O modelo do produto descreve os principais componentes do sistema e como eles estão relacionados, buscando fornecer informações sobre a arquitetura do sistema, arquitetura da aplicação e arquitetura do software. O modelo do processo, por sua vez, especifica as atividades necessárias para o desenvolvimento e implantação do sistema, incluindo a análise de requisitos, desenho, teste e implantação [2].

A próxima etapa é o plano do projeto, momento em que deve ser feito um planejamento e, se necessário, cronograma. Essa atividade pode ser apoiada por técnicas do gerenciamento de projetos utilizadas em outras áreas do conhecimento [2].

Em seguida, deve ser realizado o desenvolvimento web, contendo tanto o conteúdo da página web como a aplicação responsável pela entrega dos dados que serão exibidos para o usuário [2]. Além disso, também será necessário desenvolver e implantar a infraestrutura, composta pelo hardware e demais componentes da rede.

Por fim, após o desenvolvimento e implantação do sistema web realizado no passo anterior, será necessário prover uma manutenção contínua, seja no seu conteúdo, componentes de hardware, componentes de software ou na rede [2]. O produto desta fase pode gerar uma nova iteração do processo de desenvolvimento web, caso seja necessário promover alguma mudança no sistema.

Como o processo de desenvolvimento web possui caráter iterativo, é importante que a aplicação seja validada no fim de cada iteração [15]. A verificação do software possui o objetivo de confirmar a qualidade na implementação e evidenciar possíveis desconformidades com os requisitos do sistema.

A verificação do sistema web pode ser composta por diversas técnicas de teste, como os testes caixa preta e caixa branca. Ambas as classes de teste buscam confirmar se a aplicação atende aos requisitos, mas eles divergem quanto ao enfoque dado no teste. Os testes de caixa branca exploram questões estruturais da aplicação web, enquanto os testes caixa preta focam nas entradas (*inputs*) e saídas (*outputs*) da aplicação web [16].

Ao tratar de testes para as aplicações web, é necessário considerar que o seu funcionamento apresenta características dinâmicas, exigindo a realização de um conjunto de testes para validar aspectos variados [15]. Alguns dos critérios que podem ser utilizados para a realização do teste são: teste da página, para validar o funcionamento das páginas web; teste do link das páginas, para validar os links de cada página web; e testes de uso,

em que a navegação de cada funcionalidade é verificada.

Desta forma, os testes de aplicações web podem ser divididos em algumas categorias, conforme itens listados a seguir [2]. O acompanhamento dessas categorias facilita a organização do roteiro de testes, funcionando como um guia dos componentes que devem ser verificados.

- Compatibilidade do navegador;
- Exibição de página;
- Gerenciamento de sessão;
- Usabilidade;
- Análise de conteúdo;
- Disponibilidade;
- Backup e recuperação;
- Transações;
- Processamento de pedidos;
- Internalização;
- Procedimentos operacionais de negócio;
- Integração entre sistemas;
- Perfomance;
- *Login* e segurança.

Apesar da importância da realização de testes, muitas vezes eles não são realizados de maneira adequada, sendo executados somente quando a aplicação está apresentando falhas ou limitações [13]. Esse tipo de teste é chamado de teste reativo e não é interessante no contexto do desenvolvimento de aplicações web, já que ele espera que as dificuldades comecem a ser apresentadas para realização do teste. É desejável que a equipe de desenvolvimento realize testes proativos, assegurando o funcionamento da aplicação e assim evitando riscos, custos para ajuste de problemas e retrabalho.

Existem algumas dificuldades relacionadas ao teste de aplicações, principalmente no que se refere ao seu custo. Entretanto, os impactos gerados com a falta de realização de testes podem ser piores e gerar problemas graves na aplicação e prejuízos na imagem dos responsáveis pelo software [13].

Capítulo 3

Processos de Desenvolvimento Guiados por Teste

Neste capítulo, que possui o objetivo de apresentar conceitos relacionados aos processos de desenvolvimento guiados por testes, são apresentadas definições sobre os processos de software, dando destaque para os processos guiados por testes. Para que isso seja possível, um breve histórico é apresentado sobre o tema com o intuito de relatar o contexto da sua popularização e facilitar o entendimento acerca da sua relevância. Em seguida, os modelos de processos de software guiados por teste são descritos, apresentando características, possíveis vantagens e desvantagens.

3.1 Processos de desenvolvimento de software

Para melhor compreensão sobre as questões relacionadas aos processos de desenvolvimentos guiados por teste, inicialmente é importante conceituar o termo processo, uma vez que esse termo possui ampla utilização em vários campos de conhecimento distintos, podendo inclusive possuir significados diferentes. O termo processo está sendo empregado neste trabalho sob o ponto de vista da Engenharia de Software, representando um conjunto de atividades relacionadas que são responsáveis por transformar entradas em saídas [17]. É importante destacar que essa transformação ocorre por meio de uma sequência de atividades, ordenadas ou não, e utiliza recursos específicos durante os seus procedimentos, que dependem do tipo de atividade sendo executada. Dentro da área de conhecimento considerada, um relevante tipo de processo existente é o processo de software.

De acordo com o conceito visto anteriormente, os processos de software podem ser caracterizados como um conjunto de tarefas e atividades responsáveis por realizar a transformação de produtos de trabalho de entrada em produtos de trabalho de saída [16]. De maneira mais prática, um processo de software pode ser definido como o conjunto de ati-

vidades que resultarão na produção de um produto de software [3]. Essas atividades são executadas por engenheiros de software, profissionais encarregados de desenvolver, prover manutenção e operar os sistemas de software. Alguns exemplos de atividades executadas por esses profissionais para cumprir as suas atribuições são: levantamento de requisitos, desenho (*design*) de software, desenvolvimento, teste de software, gestão e configuração de modificações.

Ao caracterizar o processo de software como um conjunto de atividades, ações e tarefas realizadas no momento em que for necessário proceder com a criação de algum produto de trabalho, Pressman [18] diferencia cada um desses procedimentos da seguinte forma: uma atividade busca o atingimento de um objetivo mais amplo, sendo aplicada independente das características específicas do software; uma ação engloba um conjunto de tarefas para produzir um produto de trabalho principal; e uma tarefa foca em objetivos menores e bem definidos. Mesmo com terminologias específicas para os elementos que o compõe, o processo de software não deve ser tratado como uma prescrição rígida de construção do software. Na realidade, esse processo deve ser visto como uma abordagem flexível em que os seus atores devem definir quais etapas aplicar para que consigam entregar o produto satisfatoriamente.

À medida que a tecnologia evolui, as empresas buscam automatizar os processos de software como uma forma de melhorar a produtividade sem afetar a qualidade e os custos. Entretanto, essa automação ainda possui algumas restrições, considerando a diversidade existente entre os processsos de software. Essa diversidade acentua a complexidade desses processos, necessitando da utilização de senso crítico e criatividade para propor soluções, o que acaba dificultando a criação de atividades automáticas realizadas somente pela máquina [3]. Outro fator crucial que também prejudica a criação de processos de software automáticos é a diversidade existente entre as empresas que produzem softwares, seja em tamanho, modelo gerencial ou localização geográfica, por exemplo. Essas questões podem afetar consideravelmente o resultado do trabalho, já que os problemas enfocados por cada uma delas podem ser diferentes [19].

Apesar das particularidades existentes nos processos de software, existem algumas atividades básicas que são essenciais para todos os produtos, sejam eles novos sistemas que estão sendo desenvolvidos ou modificações de sistemas já existentes, sejam sistemas simples ou mais complexos. Sommervile [3] elenca como atividades fundamentais para o processo de software: especificação de software, processo de definição das funcionalidade e restrições do software; projeto e implementação de software, etapa em que o software será efetivamente produzido; validação de software, onde são feitos os testes para garantir que o produto está aderente às necessidades do cliente; e evolução do software, processo constante de transformação que o software deve passar para atender às necessidades do

cliente, que são mutáveis.

Considerando essas atividades primordiais para a construção de sistemas de software, é possível caracterizar o processo de desenvolvimento de software como um dos componentes dos processos de software. Dessa forma, o processo de desenvolvimento de software pode ser definido como um processo de conversão da necessidade de usuários em produtos de software, envolvendo etapas de requisitos, desenho, codificação e teste [17]. Esse processo de desenvolvimento de software pode ainda ser cíclico, de modo que as suas atividades devam ser novamente executadas após a conclusão do seu primeiro fluxo de execução.

Existem diversas técnicas, modelos e abordagens diferentes para caracterizar os processos de desenvolvimento de software [3]. Cada uma delas possui um enfoque distinto e busca explorar alternativas para o desenvolvimento de software, de modo que os analistas responsáveis por esse processo possam selecionar a perspectiva que esteja mais aderente às suas necessidades. Esse trabalho enfoca o processo de desenvolvimento guiado por testes.

O processo de desenvolvimento guiado por testes é uma estratégia de desenvolvimento em que os casos de teste devem ser escritos antes da codificação do programa. Com isso, a criação do software será realizada por meio de pequenas iterações, onde os casos de teste serão escritos e aplicados no programa até que possuam um resultado satisfatório [20]. Embora essa técnica tenha um grande foco sobre os testes, é importante destacar que essa não é apenas uma técnica de verificação do software [21]. Na realidade, ela é uma técnica de análise, que direciona decisões relacionadas ao desenho e programação do software. Janzen e Saiedian [20] indicam que o entendimento desse conceito é uma das maiores dificuldades no processo de desenvolvimento guiado por testes, principalmente por quem está começando a utilizar essa técnica.

Antes da exploração das caraterísticas, dificuldades e benefícios dessa técnica, inicialmente serão trabalhados os aspectos históricos do desenvolvimento guiado por testes. Essa abordagem é importante para situar em que momento no tempo ela foi popularizada, possibilitando assim uma melhor compreensão da realidade em que se encontra atualmente. Para iniciar o estudo das questões históricas dessa técnica, é relevante ter em mente que o desenvolvimento guiado por testes emergiu em conjunto com o modelo de desenvolvimento ágil de software [20].

3.2 Histórico

O desenvolvimento ágil de software é um modelo que combina diretrizes de desenvolvimento buscando priorizar aspectos relacionados à satisfação do cliente e entrega rápida e incremental do software [18]. Esse modelo foi proposto a partir da década de 1990, em um contexto onde os negócios operam em nível global e estão sujeitos a mudanças cada vez

mais rápidas [3]. Sendo assim, constatou-se a necessidade de criação de produtos de software com desenvolvimento acelerado para aproveitar oportunidades no mercado e ter uma vantagem competitiva sobre os concorrentes. Entretanto, não seria possível desenvolver e entregar produtos de software de maneira rápida em um ambiente com tantas mudanças e incertezas seguindo um modelo tradicional de desenvolvimento, baseado em especificações completas de requisitos, projetos, desenvolvimento e teste de software. Portanto, esse foi o cenário em que o modelo de desenvolvimento ágil foi proposto, com processos iterativos e intercalados de especificação, projeto, desenvolvimento e teste.

A popularização desse modelo está atrelada a um manifesto criado em 2001 durante uma reunião entre 17 desenvolvedores de software, chamado de manifesto para o desenvolvimento ágil de software [18]. O manifesto é uma declaração com alguns valores e princípios que esses desenvolvedores estabeleceram como essenciais para o desenvolvimento de software. É importante destacar que os desenvolvedores responsáveis pela criação do manifesto já praticavam metodologias mais leves como alternativas às metodologias tradicionais. Nesse sentido, o manifesto ágil foi proposto como uma nova forma para desenvolvimento de software de uma maneira melhor, sob o ponto de vista dos seus autores, priorizando aspectos relacionados a interação entre os indivíduos, colaboração com o cliente, respostas às mudanças e funcionamento do software [22]. As principais propostas do método ágil estão centradas nesses quatro valores, reconhecendo a importância de planejamento, processos, ferramentas e documentação, mas atribuindo uma carga maior aos valores de comunicação, colaboração e adaptabilidade.

Dentre as principais características do desenvolvimento ágil de software, é possível destacar os seus ciclos de desenvolvimento curtos e iterativos, equipes auto-organizadas, desenho simples do software, refatoração do código, desenvolvimento guiado por testes, grande envolvimento do cliente no projeto e criação de um produto de trabalho entregável a cada iteração do ciclo de desenvolvimento [16]. Essas características estão de acordo com os princípios do manifesto ágil, que preconizam uma mudança de cultura no que se refere ao processo de desenvolvimento de software. A Figura 3.1 exemplifica um modelo de processo iterativo genérico com o desenvolvimento incremental, metodologia utilizada no desenvolvimento ágil de software.

Considerando as suas características, é justificável concluir que o método ágil pode ser positivo em diversos contextos, pois ele adota uma abordagem de desenvolvimento de software incremental que se aproxima do modo como os seres humanos resolvem os problemas [3]. As pessoas raramente trabalham em soluções de maneira antecipada e completa. Frequentemente elas buscam soluções por meio de passos, que podem inclusive ser retrocedidos quando o indivíduo perceber que cometeu algum erro. De qualquer forma, ainda existe um grande debate acerca dos benefícios do método ágil e sua aplicabilidade.

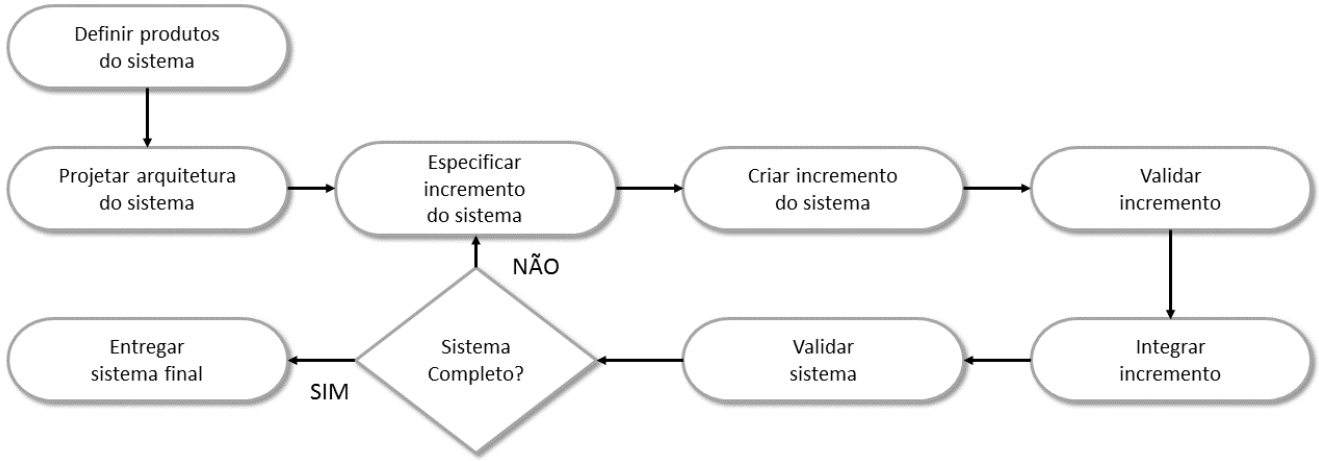

Figura 3.1: Processo de desenvolvimento iterativo (Fonte: [3]).

A entrega rápida não é o centro da discussão, mas sim como aliá-la com a necessidade dos clientes hoje e tornar os seus benefícios extensíveis para atender as necessidades dos clientes também em longo prazo [18].

É possível assumir que esse modelo de desenvolvimento propõe algumas técnicas e conceitos interessantes. No entanto, existem alguns cenários em que a sua aplicação é contestada e pode inclusive ser prejudicial. Algumas das potenciais dificuldades relacionadas ao processo de desenvolvimento ágil estão no gerenciamento, verificação, manutenção e contrato do software [3]. Esses fatores dificultadores estão associados com a inexistência de uma documentação detalhada criada durante a especificação do software para servir como insumo em outras etapas do processo de desenvolvimento do software.

Por isso, há espaço para métodos com modelos de desenvolvimento distintos do modelo ágil. Além disso, outros métodos surgem constantemente a partir da combinação entre o desenvolvimento ágil e o desenvolvimento baseado em planejamento, com o intuito de balancear as práticas utilizadas de cada um desses modelos para chegar em um processo próximo do ideal [16]. Os métodos ágeis seriam mais adequados para aplicação em sistemas de pequenas e médias empresas ou para produtos de computadores pessoais, enquanto outros métodos podem ser mais adequados para empresas maiores, com sistemas críticos, de larga escala ou que a comunicação entre os desenvolvedores seja dificultada.

Há diversos métodos ágeis descritos na literatura, como *Rapid Application Development*, *eXtreme Programming*, Scrum e *Feature-Driven Development* [16]. Esse tópico enfocará somente no método *eXtreme Programming* (XP), considerando que uma das práticas desse método é o desenvolvimento guiado por testes, um dos principais objetos de estudo deste trabalho.

XP é um popular método de desenvolvimento ágil de software. Beck [23] define cinco valores principais como a base desse método: comunicação, simplicidade, *feedback*, coragem e respeito. Sobre cada um desses valores, algumas atividades, ações e tarefas são definidas, fundamentando assim os princípios desse método.

Esse método é baseado na expressão dos requisitos como cenários, chamados de histórias do usuário, e são implementados na forma de diversas tarefas. Os programadores trabalham em duplas e, antes de começar a escrever o código, desenvolvem testes para cada tarefa estabelecida. Quando o código é então produzido, os testes escritos anteriormente precisam ser executados com sucesso para que essa funcionalidade seja integrada ao sistema [3]. A Figura 3.2 ilustra a produção de um incremento do sistema que está sendo desenvolvido por meio do XP.

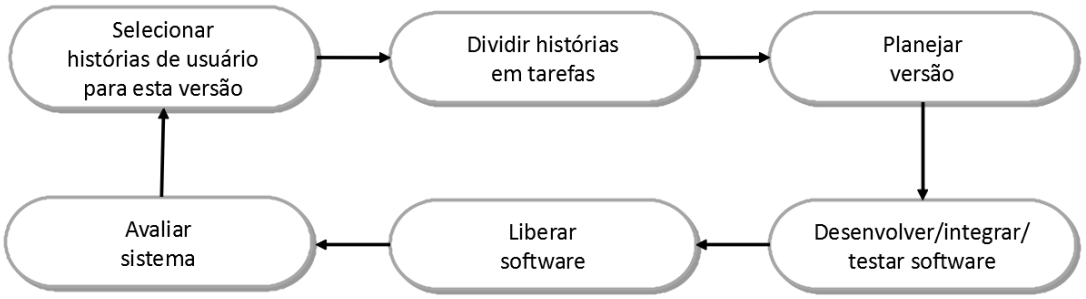

Figura 3.2: Ciclo de um *release* em XP (Fonte: [3]).

Uma prática do método XP é a simplicidade que o projeto de software deve possuir. Esse aspecto indica que qualquer complexidade no projeto de software deve ser removida, mantendo somente elementos simples. Considerando o funcionamento do fluxo de entregas incrementais apresentado, outras características marcantes do método XP que podem ser elencadas são as entregas frequentes, refatoração contínua do código e a programação em pares, que junta pessoas que podem ter visões diferentes e complementares durante o processo de desenvolvimento [23]. Entretanto, a prática considerada como uma das maiores inovações utilizadas no método XP e que será o foco deste trabalho é o desenvolvimento guiado por testes.

Tendo em vista todas as questões trabalhadas do contexto histórico em que essa prática foi proposta, as características do TDD podem ser estudadas para o entendimento da sua relevância, levantando possíveis vantagens e desvantagens no processo de desenvolvimento de software.

3.3 Desenvolvimento guiado por testes

O desenvolvimento guiado por testes também pode ser caracterizado como desenvolvimento *test-first*, *test-first programming*, *test-driven design*, *test-first design*, *test-driven development* ou com a sigla TDD [20], que será utilizada neste trabalho.

No TDD, a estratégia de testes é trabalhada desde o início do desenvolvimento do software, especificando o comportamento que a funcionalidade que está sendo desenvolvida deve possuir [3]. Porém, os testes nesse processo devem ser simples, isolados e automáticos, pois eles estão sendo utilizados somente como um ferramenta de instrumentação do desenvolvimento [23].

Essa primeira característica abordada é importante para esclarecer uma discussão frequente no contexto dessa técnica: considerando que os programadores devem escrever testes a todo o momento, o TDD deve ser visto como uma prática de verificação e teste de software ou como uma prática de análise e desenho de software, uma vez que os testes vão prover informações que os programadores utilizarão para o desenvolvimento do software [24]. Diversos autores concordam que o TDD deve ser encarado como uma prática de análise, desenho e desenvolvimento de software [20] [21] [24]. Os testes empregados nessa técnica serão instrumentos de análise, desenho e verificação se as principais especificações do programa estão funcionando.

O TDD é uma prática baseada na repetição de um ciclo de pequenas atividades. Antes de iniciar a escrita do código de alguma funcionalidade, o desenvolvedor deve escrever um teste para essa funcionalidade. Como o código ainda não foi escrito, a primeira execução desse teste vai falhar. Na sequência, o desenvolvedor deve implementar a funcionalidade desejada, de modo que o teste seja realizado com sucesso. Esse processo deve ser realizado com simplicidade, onde tanto o teste como a implementação devem ser os mais simples possíveis para o caso. No final, o código deve ser refatorado para eliminar duplicações e elementos desnecessários [25].

Segundo Turhan et al. [26], o TDD pode ser resumido nos seguintes passos:

1. Seleção de uma pequena tarefa;
2. Escrita de um teste para essa tarefa;
3. Execução de todos os testes para verificar que os novos falham;
4. Escrita do mínimo de código possível para conclusão da tarefa;
5. Execução de todos os testes para verificar se todos estão passando;
6. Refatoração do código, se for necessário;
7. Repetição do processo a partir do primeiro passo.

Esse ciclo também é chamado de "Vermelho-Verde-Refatora", como uma referência às etapas do processo de desenvolvimento, onde o vermelho está associado ao teste falhando e o verde relacionado ao teste sendo executado com sucesso [25] [27]. A Figura 3.3 também exemplifica o ciclo do TDD, deixando clara essa referência ao ciclo "Vermelho-Verde-Refatora".

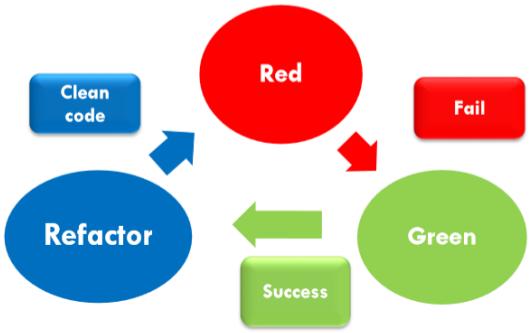

Figura 3.3: Ciclo do TDD (Fonte: [4]).

Como o início da implementação de cada tarefa prevê o desenvolvimento de um teste antes, essa prática possibilita a organização de um conjunto de testes para o sistema que podem ser executados rapidamente [3]. Com isso, todos os testes podem ser executados a cada nova entrega e isso representa uma facillidade na detecção de possíveis problemas que a nova funcionalidade introduziu no software.

Para que essa abordagem seja possível, é importante utilizar ferramentas de teste automatizadas para execução dos testes [3]. Essas ferramentas possibilitam que os testes sejam escritos como componentes que podem ser executados imediatamente e a qualquer momento do processo de desenvolvimento. Desta forma, as ferramentas de teste facilitam a criação e a execução de testes do software [20]. Existem muitos softwares comerciais e de código aberto que se propõem a atender esse propósito.

Também é importante destacar o papel adaptativo do TDD, que está relacionado com a necessidade contínua de recebimento de *feedback* durante o processo de desenvolvimento para que a qualidade do software seja avaliada e melhorada [25]. Esses *feedbacks* são recebidos pelos desenvolvedores na forma de resultado dos testes aplicados às tarefas.

Outras características dessa prática, inclusive já exploradas na seção anterior durante a abordagem do método XP, referem-se ao seu caráter iterativo e incremental [20], que podem ser claramente observados nos passos do ciclo "Vermelho-Verde-Refatora"utilizado pelo TDD.

Com base nas características e funcionamento do TDD, é possível compreender algumas vantagens e desvantagens que essa prática pode oferecer. Porém, grande parte dos estudos existentes para avaliar os efeitos do TDD possuem maior enfoque em questões

relacionadas à qualidade externa do sistema [25]. De qualquer forma, é importante trabalhar essas questões como possíveis benefícios ou malefícios da prática, considerando que existem diversas variáveis que podem influenciar no resultado de um estudo e que muitas vezes não podem ser controladas, como: ambiente em que o estudo foi realizado, experiência dos desenvolvedores com o TDD, experiência dos desenvolvedores com a linguagem de programação do sistema, tipo de software que está sendo desenvolvido, entre outros aspectos.

Uma das possíveis vantagens no uso do TDD está relacionada com o conjunto de testes que é gerado durante o desenvolvimento do software. Esses testes podem servir como aplicações práticas das especificações das funcionalidades do sistema, ajudando a evitar erros de regressão, onde novas funcionalidades podem acarretar erros em funcionalidades já existentes no sistema [25]. Os testes também são úteis para o processo de refatoração do código, garantindo a segurança desta etapa do desenvolvimento do software.

O uso do TDD também pode representar um entendimento melhor por parte dos desenvolvedores das necessidades dos clientes, pois a especificação precisa ser compreendida detalhadamente antes da implementação. Isto indica que eventuais dúvidas, omissões e ambiguidades na especificação teriam que ser esclarecidas no início do processo de desenvolvimento [3]. Outro problema que possivelmente seria evitado com o TDD é o das entregas do desenvolvedor estarem em um ritmo mais rápido e frequente que o testador.

Turhan et al. [26] fizeram uma revisão sistemática de estudos para verificar o que eles dizem a respeito dos efeitos do uso do TDD. O foco do trabalho era coletar evidências sobre os efeitos na qualidade interna do código, qualidade externa, produtividade e qualidade de teste. Após análise, concluíram que os estudos sugerem que não existe um efeito consistente sobre a qualidade interna, já que os resultados variaram muito de acordo com as métricas utilizadas (por exemplo, coesão e reuso foram métricas que apresentaram resultados distintos). No entanto, ao considerar a qualidade externa, algumas evidências sugerem que o TDD melhora essa qualidade, ainda que alguns estudos tenham sido inconclusivos para essa dimensão. Para a qualidade de teste, algumas evidências sugerem que a prática também melhora essa qualidade, o que era esperado, uma vez que essa abordagem utiliza os testes como instrumentos principais para o desenvolvimento de software. Por fim, a produtividade foi a dimensão mais controversa, pois duas linhas de abordagens opostas foram observadas. Em uma delas, foi esperado que o TDD melhorasse a produtividade devido a facilidade de troca entre tarefas simples no processo de desenvolvimento, enquanto a outra apontou que o *overhead* associado ao TDD durante a criação e execução de testes impacta negativamente na produtividade.

Tosun et al. [28] conduziram um estudo com profissionais objetivando o entendimento da eficácia dos testes unitários no TDD. Neste estudo, o TDD foi comparado com uma

abordagem incremental com teste realizado após o desenvolvimento, chamado de *Incremental Test Last Development* (ITLD), para verificar qual das práticas apresentaria o melhor resultado. O estudo leva a conclusões diferentes dependendo da métrica utilizada na análise. Em contextos específicos, o TDD auxilia na escrita de casos de teste mais completos e, em outros cenários, o ITLD acaba sendo melhor para escrita de testes unitários com maior cobertura do código fonte.

Uma pesquisa realizada por Aniche, Ferreira e Gerosa [24] procurou compreender a visão que os indivíduos que praticam TDD possuem sobre a influência desta prática no seu dia a dia. Essa pesquisa foi realizada utilizando algumas perguntas orientadoras como base, buscando estimular uma discussão entre os participantes para que eles se sentissem mais confortáveis de dar o seu ponto de vista sobre as questões. Durante as falas dos participantes, os pesquisadores conseguiram observar alguns pontos interessantes que estão de acordo com alguns fatores já relatados na literatura. A maioria das pessoas considera o TDD como uma prática importante de desenho (*design*) de software e inclusive relatam que o *feedback* constante que os testes fornecem ajuda a corrigir possíveis problemas no desenho (*design*) do software desde o início do processo. Outro consenso entre o grupo foi relacionado à facilidade da refatoração do código no TDD, considerando que os testes ajudam a apontar qualquer problema de imediato. Apesar desses benefícios relatados entre os usuários da prática, os participantes do estudo indicaram uma relutância em acreditar na prática quando foram apresentados a ela. Além disso, eles indicaram que o TDD pode ajudar e agilizar o desenho do software, mas a prática em si não resolve os problemas que esse processo possui. Esses problemas no TDD foram reforçados pelos usuários com a atribuição de um alto grau de dificuldade no aprendizado da prática e também pelo fato de que a produtividade parece ser diminuída quando a prática começa a ser utilizada.

O trabalho de Khanam e Ahsan [27] procurou levantar diversos estudos conduzidos nos últimos 10 anos para examinar os efeitos do TDD em parâmetros de qualidade de software, eficácia, velocidade de desenvolvimento, qualidade de teste, produtividade e refatoração do código. De acordo com o estudo, os desenvolvedores tendem a utilizar abordagens com teste após o desenvolvimento, até porque muitos acreditam que o TDD requer um treinamento e aprendizado rigoroso que impacta negativamente na produtividade. Além disso, alguns pesquisadores consideram que a principal deficiência do TDD é o *overhead* adicionado ao processo de desenvolvimento, o que também prejudica a produtividade. Nesse contexto, eles elencam alguns cenários nos quais não é bom utilizar o TDD, que são situações em que os seus pontos negativos seriam reforçados. Apesar de não encontrarem evidências consistentes nos estudos analisados, nem para questões negativas como para positivas, os autores levantaram alguns benefícios na aplicação do TDD, como por exemplo: códigos modulares e de fácil depuração (*debug*) e detecção de erros antecipada;

maior facilidade de manutenção e refatoração; testes funcionam como documentação; e testes fornecem *feedback* constante do sistema.

Finalmente, outro estudo exploratório foi proposto [29] com desenvolvedores de software que nunca haviam usado TDD para verificar as percepções iniciais desses profissionais sobre o efeito da prática. Como relatado em outros trabalhos, a curva de aprendizado do TDD faz com que a sua aplicação inicial seja lenta. Uma das principais dificuldades relatadas foi saber como iniciar o processo e como escrever testes para funcionalidades que ainda nem existem. De maneira geral, grande parte dos participantes notou melhora na qualidade do código e foi observada uma maior confiança na implementação das funcionalidades, provavelmente amparada pela verificação feita para garantir que estão corretas. É interessante observar também as dificuldades relatadas neste estudo para a aplicação do TDD, que variam entre a falta de cultura e habilidade, dificuldades no teste unitário e dificuldades no uso da técnica como ferramenta de desenho de software. Um aspecto relevante deste estudo também foi a abordagem de situações inesperadas relacionadas aos testes, na qual os participantes se depararam com cenários onde, por exemplo, os testes passaram quando deveriam ser rejeitados, os testes foram rejeitados quando novas funcionalidades foram criadas (problema de regressão) e a refatoração gerou erro em algum teste.

Aniche e Gerosa [30] abordam uma perspectiva diferente ao tratar dos erros mais comuns observados no TDD. Esta abordagem buscou apresentar alguns problemas listados pelos profissionais que trabalham com o TDD, possibilitando o estudo de alguns gargalos no uso da técnica. Alguns dos problemas listados foram: esquecer a refatoração do código; não começar pelo teste mais simples; executar somente o teste da funcionalidade atual; e não implementar o código mais simples que faria com que o teste fosse executado com sucesso.

Considerando os diferentes estudos realizados com o TDD, é possível observar que essa prática ainda envolve muitas incertezas. Não há um cenário específico mapeado em que a prática seja recomendada como o modelo ideal [26]. Nesse caso, o melhor a ser feito é verificar as diversas variáveis que estão associadas ao contexto em que o software será desenvolvido para avaliar se o TDD pode ser positivo. Por isso, é importante que se conheça outras práticas de desenvolvimento de software, para escolher a mais adequada para o projeto.

3.4 Alternativas ao TDD

Os trabalhos listados na seção anterior que buscaram avaliar os efeitos do TDD geralmente utilizaram características de outras abordagens para efeito de comparação. É natural uti-

lizar essa estratégia, já que uma referência muitas vezes auxilia no entendimento. No caso do TDD, a comparação com práticas que realizam o teste somente após o desenvolvimento é comum, pois elas apresentam contrastes interessantes de serem estudados. A abordagem de teste após o desenvolvimento também pode ser incremental [28], mas existe uma diferença significativa na forma como os testes são tratados em cada uma das práticas. Com o teste após o desenvolvimento, os desenvolvedores codificam o programa antes e somente depois escrevem os testes para essa funcionalidade.

Muitas vezes, a prática de teste após o desenvolvimento pode não acompanhar uma abordagem incremental de desenvolvimento de software. Na realidade, o modelo mais tradicional de processo de software é o modelo em cascata, onde as atividades de especificação, desenvolvimento, verificação e evolução são representadas como fases separadas no desenvolvimento de software [3]. Nesse modelo, as fases são bem definidas e podem inclusive ser tratadas por equipes diferentes.

Ainda que o desenvolvedor queira utilizar um modelo diferente do tradicional, existem outras práticas que podem ser consideradas, algumas até semelhantes ao TDD. Uma das alternativas é a *acceptance test-driven development* (ATDD), que representa uma variação do TDD, onde os testes de aceitação são escritos antes do desenvolvimento com base no entendimento que a equipe teve dos requisitos. Como os testes de aceitação serão escritos antes da funcionalidade, é fundamental para esse modelo que os desenvolvedores tenham uma colaboração forte com os clientes para compreensão completa dos requisitos. O ATDD auxilia na transformação dos requisitos em casos de testes de aceitação, prevendo a automação do teste de aceitação, e possibilita a verificação de funcionalidades do sistema [27].

Outro estilo de desenvolvimento que também representa variações do TDD é o *behavior driven development* (BDD). Essa abordagem acaba mesclando TDD e ATDD ao combinar testes unitários e testes de aceitação em contextos específicos [27]. Enquanto essas duas técnicas estão mais concentradas no estado do sistema e na sua frequente verificação, o BDD enfatiza o comportamento que o sistema deve possuir. Para isso, os testes são escritos utilizando uma linguagem natural, facilitando a comunicação entre os desenvolvedores e clientes para validar aspectos relacionados ao comportamento do sistema [4]. No final, o desenvolvimento acaba não sendo guiado pelos testes propriamente ditos, mas sim pelo comportamento do sistema.

Há também outras vertentes para tratar o processo de desenvolvimento de software dentro de um contexto de desenvolvimento ágil de software. Um modelo usado no contexto empresarial que está amparado no método ágil é o Scrum. Seus princípios estão baseados no manifesto ágil, assim como a XP, mas o processo de desenvolvimento de software ocorre de maneira distinta. No Scrum, é definido um *backlog* de demandas com base

nas necessidades do cliente. A partir deste *backlog* são definidas *sprints*, que consistem na organização das tarefas e entregas dos produtos de trabalho para desenvolvimento do software. Esse processo é amparado por reuniões diárias realizadas entre a equipe para alinhamento do que está sendo feito e quais impedimentos existentes [18].

No contexto de métodos ágeis, outra abordagem disponível para o desenvolvimento de software é a *feature driven development* (FDD). Assim como os demais modelos da metodologia ágil, o FDD também promove a colaboração e utiliza uma abordagem incremental. No entanto, ele considera as funcionalidades do sistema como pequenos blocos funcionais e entregáveis que podem ser implementados em duas semanas ou menos [18]. Com isso, seus fluxos de trabalho são simples e pequenos, promovendo a garantia da qualidade do sistema.

Portanto, diversas práticas podem ser usadas no desenvolvimento de software. Muitos profissionais preferem mesclar práticas diferentes com o intuito de aproveitar as qualidades que ambas podem oferecer, o que aumenta ainda mais a disponibilidade de modelos possíveis para se empregar em algum projeto. Sendo assim, é importante que os desenvolvedores conheçam um conjunto de práticas diferentes, podendo selecionar a mais adequada ao seu projeto.

Capítulo 4

Automação de Testes

Neste capítulo, que possui o objetivo de abordar o processo de automação de testes no contexto do desenvolvimento guiado por testes, são descritos conceitos relacionados à automação de testes. Sendo assim, inicialmente a definição de automação é apresentada dando enfoque para a automação do processo de teste. Posteriormente, são relacionadas algumas ferramentas que auxiliam no processo de automação de testes, caracterizando-as com base em alguns critérios específicos. Por fim, é feito um paralelo do processo de automação de testes com as aplicações web e com o processo de desenvolvimento guiado por testes, temas já apresentados nos Capítulos 2 a 3.

4.1 Definição e histórico

Para possibilitar a compreensão do significado de automação, inicialmente é importante definir esse termo e conhecer a sua origem, considerando que esse termo é utilizado em vários contextos e cenários diferentes. A automação pode ser caracterizada como a conversão de equipamentos ou processos para uma operação automática [17]. Outras definições indicam que a automação é um processo de controle automático da manufatura de produtos, o uso de meios eletrônicos e mecânicos para substituir o trabalho humano ou a aplicação de um controle automático em qualquer ramo da indústria ou ciência [31]. Parasuraman e Riley [32] definem a automação como a execução de uma função, antes realizada por um ser humano, por alguma máquina.

Apesar da existência de diversos conceitos diferentes para esse termo, é possível observar que as caracterizações apresentam grandes semelhanças. Desta forma, de maneira geral, a automação pode ser vista como a transformação de um processo para que ele passe a ser executado de forma automática, utilizando instrumentos mecânicos e tecnológicos.

O processo de automação teve uma grande popularização com a revolução industrial, onde houve um grande aumento na mecanização de processos, principalmente na indústria

automobilística [33]. Diversas empresas buscaram implantar mecanismos automáticos em suas linhas de produção, substituindo o uso de pessoas por máquinas para produção de determinados produtos. O objetivo dessa substituição era de otimizar os processos produtivos, reduzindo custos e maximizando resultados. Com a evolução tecnológica, é possível observar uma utilização mais ampla de recursos automáticos em diversos processos e produtos, tais como: automóveis, aviões, naves espaciais, dispositivos médicos, sistemas para controle de tráfego aéreo, sistemas militares, aparelhos de ar condicionado, entre outros [34].

A partir do aumento na capacidade das máquinas e desenvolvimento de novos sistemas computacionais, a automação está sendo implantada em cenários cada vez mais complexos, tentando fazer com que os computadores atuem inclusive na tomada de decisão [34]. Para que isso seja possível, é necessário fazer com que as máquinas tenham sensores para receber entradas (*inputs*), realizem o armazenamento e processamento de grandes volumes de dados e gerem resultados que possam embasar processos decisórios automáticos. Esses processos são empregados em sistemas no contexto de *big data*, inteligência articial e robótica, por exemplo.

Mesmo com a automação desempenhando um papel importante para auxiliar os seres humanos em diversas atividades, inclusive substituindo-os completamente em alguns trabalhos braçais, a automação de atividades cognitivas ainda é muito complexa e de difícil realização [32]. Por isso, mesmo com a crescente automação de algumas funções, as pessoas continuam sendo necessárias para desempenho de algumas atividades, seja programando o nível de automação implementado, fornecendo entradas (*inputs*) para o processo ou analisando os resultados gerados. Portanto, a automação acaba afetando o tipo de trabalho realizado pelas pessoas, que tende a ser menos físico e braçal, pois as atividades mecânicas são mais fáceis de serem automatizadas, e passa a ser mais mental e cognitivo [33].

É importante ter em mente que a automação pode ser aplicada até mesmo em parte de uma atividade, de modo que os demais processos permaneçam sendo tratados de forma não automática. Essa característica implica que a automação pode ser aplicada de maneira flexível, variando em diferentes níveis crescentes dentro de um processo, que vão desde a não existência de recursos automáticos até a automação completa do processo. Baseando-se nessa definição, Parasuraman, Sheridan e Wickens [31] sugeriram a criação de uma escala com 10 níveis crescentes de automação, no qual o computador nao oferece nenhuma assistência no primeiro nível e as ações devem ser todas realizadas pelas pessoas e, no último nível, o computador age de maneira autônoma, decidindo tudo e não depende de nenhuma intervenção humana. Por fim, no contexto do desenvolvimento de software, um dos processos que pode, em grande parte, ser automatizado é o processo de teste de

software.

4.2 Automação de testes

Um processo importante no desenvolvimento de software é o teste de software, que consiste na verificação dinâmica de que o programa está se comportando conforme esperado para um conjunto finito de casos de teste [16]. O conjunto de casos de teste que serão considerados para verificação do software dependem do propósito de realização do teste e, por isso, devem ser definidos em consonância com critérios específicos do teste a ser realizado. A efetividade do teste deve ser avaliada com base nos resultados gerados após o processamento do teste.

Como os testes possuem grande relevância na garantia da qualidade do software, esse processo tem sido aplicado em diversas fases do ciclo de desenvolvimento do software. Nesse contexto, existem diversas fases distintas de teste, que podem variar de acordo com o escopo ou objetivo do teste [16].

Já que os testes de software geralmente envolvem custos altos para a sua realização e são processos importantes para determinação da qualidade do software, muitas pessoas recorrem à sua automação para minimizar custos e melhorar os resultados. A automação de testes consiste na execução de atividades de teste de maneira automática, por meio da criação de roteiros e *scripts* de teste para verificação da aderência do sistema aos requisitos de teste [35]. Essa automação geralmente é realizada com o uso de ferramentas de automação de testes.

Um motivo para a automação de testes é o fato de que a realização de testes manuais pode consumir tempo significativo. Além disso, os testes automáticos aumentam a eficiência, devido a possibilidade de execução de casos de teste de maneira iterativa com uma maior facilidade e rapidez [36]. Alguns tipos específicos de teste, como por exemplo os testes de performance e teste de estresse (*stress*), dificilmente conseguem ser realizados manualmente sem o auxílio da automação. Portanto, a automação de testes tende a agilizar as atividades envolvidas na realização de testes de software. Essa questão é muito importante, pois quanto mais cedo os erros do ciclo de desenvolvimento de software são encontrados, mais fácil, rápida e menos custosa é a sua correção [35].

Entretanto, ainda que a automação apresente retornos positivos, é fundamental ter em mente que os seus benefícios geralmente são atingidos a médio e longo prazo. Isso implica que a automação de testes não vai gerar uma imediata redução de esforço e diminuição da carga de trabalho manual. Na realidade, o processo de implantação da automação de testes pode, em um primeiro momento, gerar uma carga maior de trabalho, considerando que ele geralmente está associado ao uso de ferramentas que precisam ser aprendidas pela

equipe e também depende de um planejamento e geração de entradas (*inputs*) para que o processo automático possa ser executado [35].

Existem algumas desvantagens na automação de testes que devem ser consideradas. De acordo com Karhu et al. [36], a principal desvantagem da automação de testes está nos custos atrelados à sua implementação, manutenção e treinamento. O custo de implementação está relacionado com o investimento realizado para automação dos testes, que variam desde a aquisição de ferramentas, contratação de pessoal especializado e tempo despendido no processo. Os custos de manutenção referem-se os recursos gastos para manter e atualizar os roteiros de teste. Por fim, os custos de treinamentos são necessários para que as pessoas tenham conhecimento do funcionamento das ferramentas de automação dos testes, promovendo a mudança das suas práticas de trabalho.

Portanto, antes de se optar pela automação dos testes do software, é preciso avaliar se a automação é aplicável ao contexto em que os testes serão realizados. Existe a expectativa de que a automação elimine a necessidade de intervenção humana, mas isso nem sempre é possível, principalmente nas etapas que dependem de análises críticas, como o planejamento e tomada de decisão [35]. A automação pode desempenhar um importante papel na realização de testes que requerem muitas repetições de uma carga de trabalho, trabalhos que são muitos lentos para execução operacional e trabalhos que críticos que são mais seguros de serem feitos com o uso de uma ferramenta de apoio [37].

A automação pode ser aplicada a diversas fases diferentes de teste. No entanto, como cada uma das fases possui enfoques e objetivos específicos, muitas vezes os testes aplicados a uma fase são diferentes das demais. Desta forma, a equipe responsável pelos testes deve definir o que será testado e quais são os resultados esperados, para que as entradas (*inputs*) necessárias sejam devidamente fornecidas e os roteiros de teste corretos sejam aplicados. Conforme apresentado por Dustin, Rashka e Paul [35], seguem as fases de teste que podem ser automatizadas:

- **Teste unitário:** existem diversas ferramentas disponíveis para realização de testes unitários do software de maneira automática. Como esses testes são muito utilizados ainda durante o desenvolvimento do software para validar uma unidade específica do código, muitas ferramentas de desenvolvimento também já possuem funcionalidades para realização de testes unitários. De qualquer forma, eles ainda dependem de algumas interações manuais, principalmente no fornecimento de entradas (*inputs*) e avaliação dos resultados.
- **Teste de integração:** esses testes, que enfocam interações entre os componentes do software, também podem ter o seu processamento automatizado. Os *scripts* criados para os seus roteiros de teste podem inclusive ser aproveitados em teste de sistema.

- **Teste de sistema:** considerando que é um teste mais abrangente, roteiros de teste já automatizados pelo teste unitário e teste de integração podem eventualmente ser reaproveitados para o teste de sistema.

Com base nessas informações, é necessária uma análise preliminar do software para verificar a aplicabilidade da automação dos testes e como ela deve ser realizada, uma vez que o seu uso incorreto pode levar ao aumento de custos sem trazer vantagens significativas. Se a automação de testes for utilizada adequadamente, esse processo pode representar uma melhora nos níveis de qualidade do software. Porém, a sua implementação constitui um desafio, já que não existem instruções muito bem definidas sobre como implementar a automação de testes de software [37]. A implementação da automação de testes pode ser facilitada por meio do uso de ferramentas criadas especificamente com essa finalidade.

4.3 Ferramentas para automação de testes

As ferramentas de teste desempenham um importante papel no processo de automação de testes, uma vez que elas são responsáveis pela execução iterativa dos testes do software. Já existem diversas ferramentas disponíveis no mercado para uso e, à medida que a tecnologia evolui, novas ferramentas são desenvolvidas. Cada uma delas é responsável pela automação de fases de teste específicas, não havendo uma única ferramenta que faça a automação completa do teste de software [7]. Por isso, é importante verificar as características das ferramentas e o contexto em que ela será utilizada para definir qual a solução mais adequada para o software que será testado.

Uma das primeiras questões que devem ser avaliadas durante a definição pelo uso de ferramentas de automação de testes é se a ferramenta utilizada será proprietária, de código aberto ou desenvolvida pela própria equipe. Para avaliar as vantagens e desvantagens de cada alternativa, Hass [7] apresentou a Tabela 4.1 com as características das classes de software.

Outras características que devem ser consideradas para definição das ferramentas de automação de teste são os seus *frameworks* e a linguagem de programação compatível com a ferramenta. Os *frameworks* estão entre as tecnologias mais utilizadas para a automação de testes e Polo et al. [8] relacionam alguns exemplos dessas tecnologias conforme itens a seguir.

- **Java:** *frameworks* JUnit (código aberto), JTest (proprietário) e JMock (livre).
- **JavaScript:** *frameworks* DOH (código aberto), QUnit (livre) e JSTest.Net (livre).
- **C/C++:** *frameworks* C++ test (proprietário), Cantata++ (proprietário), Opmock (licença pública geral) e Google C++ Testing *Framework* (livre).

Tabela 4.1: Características de classes de ferramentas de automação de testes de acordo com a sua licença (Fonte: [7]).

Proprietária	**Código aberto**	**Desenvolvimento próprio**
Alguns aspectos da ferramenta e do processo geralmente precisam ser adequados	A ferramenta pode ser alterada e as melhorias devem ser compartilhadas	A ferramenta pode ser desenvolvida de acordo com as necessidades da empresa
O preço geralmente é fácil de ser calculado	A ferramenta é livre, mas podem haver taxas de licenciamento	O custo envolvido pode ser complexo de ser estimado
Geralmente, o pagamento deve ser feito em um curto espaço de tempo	Não existem tarifas imediatas a serem pagas	O desenvolvimento e, consequentemente, o pagamento podem ser feitos no ritmo da empresa
Atende ao seu propósito da melhor forma possível	A qualidade depende do histórico de uso e aceitação da ferramenta pela comunidade	A qualidade depende do seu desenvolvimento, podendo apresentar o resultado mais aderente à empresa

- **.NET:** *frameworks* NUnit (livre), DbUnit.NET (código aberto), MbUnit (livre) e QuickUnit.NET (proprietário).
- **PHP:** *frameworks* PHPUnit (código aberto), Apache-test (código aberto) e Entrance PHP (proprietário).
- **Aplicações Web:** *frameworks* HtmlUnit (código aberto) e Selenium (código aberto).

Uma capacidade frequentemente encontrada em ferramentas de automação de teste é a captura das interações do usuário com o software, possibilitando a re-execução dos comandos de maneira automática. Essa capacidade é interessante para as ferramentas de automação, pois facilita a criação dos roteiros de teste, que não precisam ser desenvolvidos na forma de um *script*, sendo gerados automaticamente pela ferramenta [8]. A Tabela 4.2 apresenta listagem de ferramentas de captura e re-execução de testes.

Ao optar pelo uso de alguma ferramenta, é importante ter consciência de que o seu uso envolve gastos com a seleção, aquisição, licenças, implementação, treinamento e manutenção. Conforme explicitado na seção anterior, esses custos representam uma desvantagem na automação de testes. Em alguns casos esse custo é direto, como na aquisição da ferramenta, mas também pode ser um custo indireto, como o tempo gasto pelos funcionários para o aprendizado da ferramenta. Como esse processo engloba aspectos de difícil mensuração, como os custos indiretos e aspectos de qualidade, a avaliação do custo-benefício da automação de testes é complexa. Com a maturidade da automação de testes ao longo

Tabela 4.2: Ferramentas de captura e re-execução de testes (Fonte: [8]).

Ferramenta	**Descrição**	**Licença**
TestCover	Serviço online de teste combinatório	Proprietário
AETG	Aplicação web que combina testes de dados	Proprietário
CTEXL	Aplicação com interface gráfica para combinação de valores	Livre
Intelligent Test Case Handler	Aplicação baseada na IDE Eclipse para geração de combinações de testes	Proprietário
AllPairs	Aplicação de linha de comando	Livre
CombTestWeb	Aplicação web que implementa diversas combinações de algortimos	Livre

do tempo, a tendência é que os custos diminuam e que os benefícios da automação fiquem mais evidentes [7].

4.4 Automação e TDD no contexto de aplicações web

Conforme apresentado no Capítulo 3, o TDD é uma técnica de desenvolvimento de software que está baseada na repetição do ciclo de algumas atividades. Essa repetição é pautada na escrita dos casos de teste de uma funcionalidade antes do seu desenvolvimento, de modo que esses testes sejam executados continuamente até que os resultados da verificação sejam bem sucedidos, após o término do desenvolvimento da funcionalidade. Como essa prática é baseada em um ciclo de repetição, é importante trabalhar com a automação dos seus passos para acelerar o processo de desenvolvimento de software.

A automação de testes executáveis para utilização no TDD pode representar uma maior agilidade no *feedback* das verificações do software, quando comparada com a execução manual dos casos de teste [5]. Além disso, a automação também pode representar uma maior completude dos cenários de teste que serão validados, considerando que mais casos de teste poderiam ser executados. O ciclo de aceitação do TDD, utilizando a automação de testes, pode ser resumido por meio da Figura 4.1.

O primeiro passo deste ciclo consiste na seleção de uma história, que são conjuntos de atividades definidas com base na especificação do software. O segundo passo é a escrita dos testes para essa história, considerando que o TDD prioriza a definição dos testes antes do desenvolvimento. O próximo passo deve ser a automação dos testes, o que facilitaria a sua execução de maneira contínua e integrada. O último passo é a implementação da funcionalidade selecionada, que será concluída quando todos os testes escritos passarem com sucesso e a refatoração do código for concluída. Após a conclusão, todo o ciclo pode ser executado novamente, selecionando-se uma nova história [5].

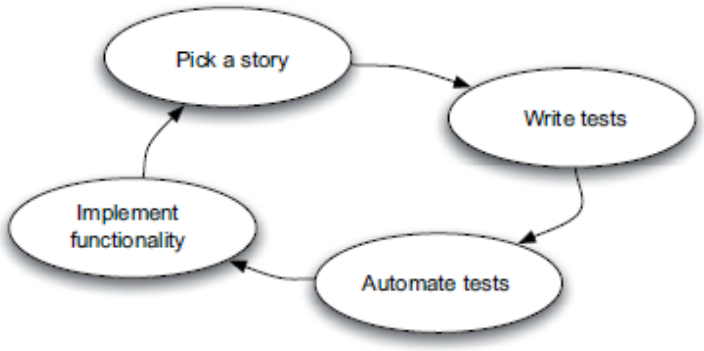

Figura 4.1: Ciclo de aceitação do TDD (Fonte: [5]).

Durante a implementação da automação do TDD, é fundamental utilizar ferramentas para auxiliar na automação dos atividades, garantindo a qualidade do processo de desenvolvimento. Algumas ferramentas já apresentadas na seção anterior podem ser utilizadas para prover essa automação, sempre observando quais os critérios que desejam ser contemplados nos testes para selecionar a ferramenta que melhor se enquadra nesses quesitos. Algumas das características que devem ser observadas para automação do TDD são: realização de testes unitários, *frameworks* para testes de aceitação, integração contínua e refatoração do código [5].

Com relação à estratégia utilizada para automação, as técnicas que podem ser utilizadas variam bastante, sendo que algumas das mais utilizadas são: ferramentas de captura e re-execução, técnicas de *scripts* de teste e geradores de casos de teste [11]. Sendo assim, o uso da automação de testes no contexto do TDD pode representar diversos benefícios para o processo de desenvolvimento do software, combinando as vantagens existentes na utilização do TDD com os pontos positivos relacionados à automação.

Considerando as informações apresentadas sobre as aplicações web, é possível notar que o seu desenvolvimento apresenta características desafiadoras, tendo em vista que ele possui ciclos de desenvolvimento rápidos, com requisições de mudança e novas implementações ocorrendo em um curto espaço de tempo [38]. Nesse contexto, a abordagem ágil é uma alternativa interessante para ser utilizada, já que trabalha com algumas práticas mais aderentes ao cenário apresentado. Uma das técnicas que pode ser utilizada é o TDD.

No entanto, a automação de testes não é trivial e representa uma atividade de grande complexidade do processo de desenvolvimento de software. Essa condição é ainda mais complicada para o TDD, considerando que ele não pode utilizar as ferramentas de captura e re-execução de testes, que é um dos instrumentos de automação mais simples e efetivos que existe. A impossibilidade de uso desse recurso é justificável porque ele é baseado na gravação da interação do usuário com o sistema para depois repetir as ações, mas o

sistema ainda não está implementado no momento da criação dos casos de testes com o TDD [6].

Nesse contexto, a seleção da ferramenta de automação de testes adequada é importante para que os testes possam ser criados com eficiência. No entanto, a escolha da ferramenta certa não é uma atividade trivial, pois diversos critérios diferentes devem ser observados sobre o projeto e a ferramenta, procurando verificar a aderência entre ambos.

Capítulo 5

Avaliação de Ferramentas de Automação de Teste

Neste capítulo, que possui o objetivo de abordar os aspectos relacionados a avaliação das ferramentas de automação de teste, são elencadas algumas informações que facilitam a análise desses softwares de teste. Sendo assim, inicialmente são descritos alguns aspectos que ressaltam a importância da avaliação de ferramentas. Em seguida, são levantados alguns critérios para avaliação de ferramentas, com enfoque para as ferramentas de automação de testes.

5.1 Avaliação de ferramentas de software

A avaliação de algo pode ser definida como a determinação do mérito, validade e valor do objeto de avaliação [39]. Sendo assim, a avaliação é um processo de juízo de valor e o seu resultado depende muito de como ele é realizado.

O processo de avaliação pode ser aplicado a diferentes domínios, tais como: avaliação de performance, avaliação de produto, avaliação pessoal, avaliação de proposta e avaliação de programa, por exemplo. Cada um desses domínios possui objetos de análise diferentes, podendo ser aplicado a praticamente todas as coisas, como: comportamente humano, times de futebol, produtos de comida etc [39].

Portanto, para que a avaliação possa ser realizada de maneira satisfatória, é necessário estabelecer as características que serão observadas, já que diferentes parâmetros podem levar a conclusões diferentes. Scriven [39] estabelece sete critérios distintos que podem ser utilizados durante um processo de avaliação, conforme itens a seguir:

- **Preferências pessoais:** o processo de avaliação pode ser baseado nas percepções do indivíduo, apesar desse critério de avaliação possuir menos importância em um

contexto profissional ou coletivo.

- **Valor de mercado:** esse critério de avaliação está baseado em um aspecto quantitativo, que é o valor coletivo atribuído ao objeto de avaliação.
- **Valor real:** esse critério de avaliação está baseado em um fator abstrato, que é o mérito definido para o objeto de avaliação, devendo ser embasado em fatos e valores para determinar seu juízo de valor.
- **Valor público:** esse critério de avaliação também está baseado em um aspecto quantitativo, que é o valor que determinado grupo atribui ao objeto de avaliação.
- **Padrões:** esse critério de avaliação está baseado em aspectos que são definidos por meio de valores públicos e se tornam padrões.
- **Valor contextual:** esse critério de avaliação está baseado em contextos específicos que devem ser considerandos para a avaliação.
- **Valor ilustrativo:** esse critério de avaliação está embasado em um modelo que é utilizado como parâmetro para comparação.

No contexto das ferramentas de software, a avaliação é importante devido ao crescimento no uso dos sistemas computacionais, que gerou um aumento no desenvolvimento e distribuição de produtos de software. A importância da avaliação de software está diretamente ligada com a capacidade de promover a qualidade dos produtos de software, seguindo padrões de específicos de qualidade, e selecionando produtos bons e aderentes a sua realidade [40]. Sendo assim, considerando a ampla oferta de produtos de softwares, a avaliação e seleção de uma ferramenta tem sido mais difícil para os consumidores, tendo em vista a existência de variadas opções disponíveis com funções semelhantes. Essa dificuldade está relacionada com a falta de padrões para análise de produtos semelhantes, diferença de performance entre produtos e a falta de conhecimento e experiência das pessoas para realizar uma escolha adequada [41].

A avaliação de ferramentas de software pode ser considerada como um processo de tomada de decisão com múltiplos critérios [42]. Esse tipo de processo decisório envolve a escolha de alternativas com base em múltiplos atributos que geralmente são conflitantes. O seu objetivo é o de ajudar os tomadores de decisão a escolherem a melhor alternativa, indicando alternativas que podem ser interessantes para o contexto estudado.

Jadhav e Sonar [42] realizaram um estudo bibliográfico da literatura associada a avaliação e seleção de ferramentas de software. Com base na metodologia utilizada, eles identificaram que as atividades mais comuns envolvendo a avaliação de software são: identificação dos critérios de avaliação, determinação de pesos para cada critério, determinação de uma

escala para cada critério, cálculo da pontuação e ranqueamento dos softwares. Apesar da importância dos critérios de avaliação e do fato de que muitos estudos utilizam opções em comum, não existe uma lista padrão de critérios para ser seguida no processo de avaliação. Além disso, a divergência existente na significação de cada critério, que geralmente está aberta à interpretação do avaliador, pode levar a uma ambiguidade, ocasionando em problemas na avaliação do software. Por isso, se faz necessário estabelecer os parâmetros de avaliação que serão considerados durante o processo.

5.2 Critérios para avaliação de software

Para solucionar o problema da inexistência de uma lista genérica com os critérios de avaliação de software e da falta de clareza e definição desses critérios, Jadhav e Sonar [42] criaram uma relação genérica com os principais critérios de avaliação que podem ser utilizados para análise de qualquer tipo de software. Os critérios foram separados em listas diferentes, de acordo com o seu propósito, conforme itens listados abaixo:

1. **Critérios relacionados às características funcionais do software**
    - **Funcionalidade incluída:** áreas ou funções da organização que o software deve atender.
    - **Objetivo principal:** áreas funcionais para o qual o software é orientado ou forte.
    - **Completude:** definida como o grau em que o software atende aos requisitos.
    - **Adaptabilidade:** nível possível de customização gerais e específicas.
    - **Abertura:** nível de abertura à desenvolvimentos adicionais (internos e externos) e a outras aplicações existentes.
    - **Interoperabilidade:** capacidade de integrar com outras ferramentas e aplicações.
    - **Níveis de segurança:** políticas de segurança suportadas pelo software (identificação do usuário, auditoria, criptografia de dados).
    - **Números de usuários simultâneos:** número de usuários simultâneos que podem utilizar o sistema.
2. **Critérios relacionados às características de qualidade do software**
    - **Soluções verticais:** número de versões customizadas do software para atender requisitos típicos ou específicos.

- **Campos customizáveis:** capacidade de personalizar o *layout* do software.
- **Relatórios customizáveis:** capacidade de personalizar o *layout* dos relatórios produzidos pelo software.
- **Tipo de interface:** tipo de interface do software.
- **Linguagens de programação:** capacidade de personalizar os módulos por linguagens de programação.
- **Middleware:** middleware padrões que são suportados pelo software (CORBA, DCOM, RMI, ODBC etc.).
- **Bancos de dados:** sistema gerenciador de banco de dados que podem ser acessados pelo software (SQL Server, Oracle, DB2 etc.).
- **Comunicação:** padrões de troca de dados que são suportados pelo software (EDI, XML etc.).
- **Variedade da plataforma:** capacidade do software de ser executado em diferentes plataformas.
- **Número de módulos:** tamanho médio de unidades de códigos independentes.
- **Número de módulos instaláveis independentes:** nível de independência entre os módulos.
- **Número de estações de trabalho:** número máximo de usuários que podem ser suportados.
- **Número máximo de camadas de distribuição:** capacidade de divisão do software entre aplicações separadas que podem ser distribuídas em diferentes servidores.
- **Número de módulos que podem ser instalados em servidores separados:** capacidade de distribuir módulos em diferentes servidores.
- **Escalabilidade:** capacidade do software de suportar um aumento no número de usuários e maior carga de transações.
- **Interface do usuário:** facilidade com a qual cada usuário pode usar a interface do software.
- **Tipos de usuário:** capacidade do software de suporte aos usuários iniciantes, intermediários e avançados ou a combinação desses usuários.
- **Visualização de dados:** capacidade do software de apresentar os dados de forma efetiva.
- **Relato de erros:** capacidade de exibição de relatórios e mensagens de erros do software.

- **Variedade de domínio:** capacidade do software de ser usado em diferentes indústrias para resolver diferentes tipos de problemas.
- **Facilidade de uso:** facilidade com a qual o usuário pode aprender e operar o software.
- **Robustez:** capacidade do software de executar de forma consistente, sem apresentar erros.
- **Backup e recuperação:** capacidade do software de suportar funcionalidade de backup e recuperação.
- **Comportamento do tempo:** capacidade do software de produzir resultados em um período de tempo razoável relacionado com a quantidade de dados.

3. **Critérios relacionados ao fornecedor**

- **Manual do usuário:** disponibilidade de manual do usuário com índice, com informações importantes e principais comandos.
- **Tutorial:** disponibilidade tutorial para aprender como usar o software.
- **Guia de solução de problemas:** disponibilidade de guia de solução de problemas.
- **Treinamento:** disponibilidade de cursos de treinamento para aprendizado do software.
- **Manutenção e atualização:** suporte do fornecedor para atualização e manutenção do software.
- **Consultoria:** disponibilidade de suporte técnico e consultoria pelo fornecedor.
- **Comunicação:** comunicação entre usuário e fornecedor.
- **Demonstração:** disponibilidade de demonstrações e versões experimentais gratuitas.
- **Número de instalações:** número de instalações do software.
- **Tempo de resposta:** nível de serviço prestado pelo fornecedor.
- **Experiência:** experiência do fornecedor sobre o desenvolvimento de produtos de software.
- **Histórico do produto:** popularidade do produto do fornecedor no mercado.
- **Popularidade do fornecedor:** popularidade do fornecedor no mercado.
- **Habilidades técnicas e negociais:** habilidades técnicas e negociais do fornecedor.

- **Experiência negocial passada:** experiência de negócios passada com o fornecedor, se alguma.
- **Referências:** número de referências dos consumidores existentes utilizando o produto.

4. **Critérios relacionados aos custos e benefícios**

   - **Custo da licença:** custo de licença do produto nos termos de número de usuários.
   - **Custo de treinamento:** custo de treinamento para os usuários do sistema.
   - **Custo de instalação e implementação:** custo de instalação e manutenção do produto.
   - **Custo de manutenção:** custo de manutenção do produto.
   - **Custo de atualização:** custo de atualização do produto quando uma nova versão é lançada.
   - **Custo do hardware:** custo do maquinário utilizado para suporte ao sistema, incluindo processador, memória e terminais.
   - **Benefícios diretos:** economias tangíveis em trabalho e equipamento, redução no custo de processamento por unidade e eliminação de cobranças de serviços externos.
   - **Benefícios indiretos:** melhoria no serviço do consumidor e tempo mais rápido de resposta de processamento.

5. **Critérios relacionados ao hardware e software**

   - **Memória interna:** armazenamento primário necessário para execução do software.
   - **Protocolos de comunicação:** protocolocos de comunicação suportados pelo software.
   - **Armazenamento externo:** armazenamento secundário necessário na forma de espaço em disco e outros dispositivos de armazenamento.
   - **Compatibilidade:** compatibilidade com hardware e software existentes.
   - **Código fonte:** disponibilidade do código fonte.
   - **Plataforma de hardware:** plataforma de hardware requerida para executar o software.
   - **Configuração de rede:** tecnologia de rede necessária para execução do software.

6. **Critérios relacionados à opinião de fontes técnicas e não técnicas**

   - **Opiniões de fontes técnicas:** opiniões sobre o software de potenciais fornecedores, representantes de vendas, especialistas, consultores externos e revistas de computação.
   - **Opiniões de fontes não técnicas:** opiniões sobre o software de subordinados, usuários finais e conhecidos externos.

7. **Critérios relacionados às saídas**

   - **Relatórios:** facilidade na geração de relatórios padrões e customizados.
   - **Entrega:** geração de arquivo, impressão ou integração com outro software.

Os critérios descritos nos itens 1 e 2 são relacionados aos aspectos de qualidade do software, enquanto os demais itens se referem à questões ligadas aos custos, benefícios, fornecedores, opiniões, hardware e software. Essa relação de critérios busca resolver o problema de que não existe nenhuma listagem padrão dos critérios de avaliação de software [42].

Um dos tipos de ferramentas de software existentes são as ferramentas utilizadas para automação de testes, que possuem uma grande diversidade de opções distintas, cada uma delas possuindo integrações diferentes com determinadas linguagens de programação, sistemas operacionais e também possuindo objetivos diferentes. Nesse caso, é importante determinar critérios de avaliação específicos para esses softwares.

5.3 Critérios para avaliação de ferramentas de automação de testes

Considerando a diversidade das ferramentas de automação de testes disponíveis e as diferenças entre as funcionalidades e o propósito de cada uma delas, é preciso estabelecer critérios específicos de avaliação das ferramentas para avaliar se ela pode ser utilizada para os testes do seu sistema.

A seleção da ferramenta de automação de testes correta é relevante para que o projeto de teste seja bem sucedido, o que depende de uma análise preliminar de diversos fatores, que incluem características do software que será testado assim como da ferramenta de automação de testes. Segundo Bajaj [43], alguns dos parâmetros mais importantes para avaliação de ferramentas de automação de testes são: facilidade de adoção, facilidade de criação de *scripts* e relatórios e o uso de ferramentas. No parâmetro facilidade de adoção, os critérios mapeados para análise são: custo da licença e facilidade de suporte. Para a

facilidade de criação de *scripts* e relatórios, os critérios são: tempo de criação de *scripts*, linguagem dos *scripts*, reconhecimento de objetos, tempo de aprendizado, velocidade da execução de *scripts*, *framework* e integração contínua. Por fim, no uso de ferramentas os critérios elencados para verificação são: suporte de aplicações que não são executadas no navegador, sistemas operacionais suportados, navegadores suportados e dispositivos suportados.

Outra abordagem para determinar critérios de qualidade para as ferramentas de automação de testes, adotado por Illes, Herrmann, Paech e Rückert [44], utiliza o padrão ISO/IEC 9126. Esse padrão define um *framework* para avaliação da qualidade do software com base em seis características: funcionalidade, confiabilidade, usabilidade, eficiência, manutenção e portabilidade. Além desses critérios, os autores incluíram outros três critérios relacionados às qualificações do fornecedor da ferramenta de automação: qualificações gerais do fornecedor, suporte do fornecedor e preço e licenciamento.

Jureczo e Mlynarski [6] utilizaram outra estratégia para estabelecer alguns critérios e avaliar quatro ferramentas de automação de testes distintas durante o desenvolvimento de parte de uma aplicação web utilizando o TDD. Os critérios de avaliação foram elaborados com base na literatura sobre o TDD e na observação dos autores, sendo estabelecidos a partir de perguntas sobre as funcionalidades e tecnologias suportadas pela ferramenta e suas respectivas opções de resposta, conforme itens listados a seguir.

1. **TDD**
    - **Pergunta:** a ferramenta de automação pode ser usada antes ou depois da implementação do código?
    - **Respostas:** fácil abordagem do TDD, difícil abordagem do TDD ou não é possível utilizar o TDD.
2. **Sistema a ser testado**
    - **Pergunta:** a ferramenta de automação pode ser usada para aplicações web?
    - **Respostas:** sim ou não.
3. **Ambiente de teste**
    - **Pergunta:** os casos de teste criados podem ser executados em um ambiente de usuário?
    - **Respostas:** execução em ambiente de usuário ou execução em ambiente de desenvolvimento.
4. **Integração de desenvolvimento**

- **Pergunta:** a ferramenta de automação pode ser integrada ao ambiente de desenvolvimento?
- **Respostas:** possível ou não possível.

5. **Implementação do teste**
   - **Pergunta:** como os casos de teste para a interface gráfica são criados?
   - **Respostas:** codificação ou captura e *replay*.
6. **Interface gráfica do usuário**
   - **Pergunta:** a ferramenta de automação utiliza identificadores ou posições baseadas em coordenadas para os elementos da interface gráfica?
   - **Respostas:** identificadores ou posições baseadas em coordenadas.
7. **Interação com banco de dados**
   - **Pergunta:** a ferramenta de automação pode interagir com o banco de dados?
   - **Respostas:** interação com o banco de dados possível, interação com o banco de dados possível por meio da integração com outras ferramentas ou interação com o banco de dados não possível.
8. **Checagem de erros**
   - **Pergunta:** a ferramenta de automação pode acessar e verificar as mensagens de erro?
   - **Respostas:** checagem de erros possível ou checagem de erros não possível.
9. **Legibilidade dos casos de teste**
   - **Pergunta:** os casos de teste são legíveis para pessoas que não são engenheiros de software?
   - **Respostas:** legíveis, difícil legibilidade ou não legíveis.
10. **Inclusão do cliente**
    - **Pergunta:** o cliente pode ser envolvido na especificação dos casos de teste?
    - **Respostas:** cliente pode ser envolvido ou cliente não pode ser envolvido.

Para cada pergunta, uma das opções de resposta foi selecionada com o uso da ferramenta de automação para que a avaliação pudesse ser realizada. Dentro do cenário criado pelo estudo, a primeira opção de resposta pré-definida era a que iria garantir uma melhor

avaliação da ferramenta. Portanto, quanto mais respostas com as primeiras opções para as perguntas, melhor seria a avaliação da ferramenta para o contexto do teste realizado [6].

Nesse estudo [6], as ferramentas avaliadas foram JFCUnit, Selenium, FitNesse, Autolt e Proven!, sendo que todas foram verificadas durante o desenvolvimento de uma funcionalidade de um sistema para mudança de preço de um produto, implementada utilizando o *framework* Spring para aplicações web.

Durante a análise das ferramentas, foi observado que o Autolt e JFCUnit não podem ser usados para testar aplicações web, além de ser muito complicado de utilizá-las com o TDD. Essas ferramentas devem ser usadas para alguns tipos mais específicos de aplicação e são mais voltadas para o uso pelos desenvolvedores [6].

O Selenium é uma ferramenta mais flexível, que possui bom funcionamento nas aplicações web, podendo ter os seus testes gerados por meio de captura e convertido para linguagens de programação [6]. Seu funcionamento já possibilita o uso em um contexto de TDD, mas ainda é um pouco complicado para usuários que não são técnicos.

Uma outra ferramenta trabalhada por Jureczko e Mlynarski [6], o FitNesse, também facilita a automação de testes para o TDD e pode ser utilizada com aplicações web. Como o seu funcionamento é mais simplificado, ele possibilita a criação de um *framework* de teste interessante e que pode ser utilizado até mesmo por usuários que não possuem conhecimentos técnicos. Um de seus problemas é que ele verifica apenas a lógica do programa, não fazendo interações com a interface gráfica, o que impede o teste de elementos da interface como as mensagens de erro.

Por fim, o Proven! foi a última ferramenta testada no contexto do estudo, sendo a única proprietária utilizada. Os seus testes são definidos utilizando HTML e são construidos de forma tão simples que os usuários sem conhecimentos técnicos podem ser participantes ativos na sua especificação. Portanto, essa ferramenta trabalha com aplicações web e possibilita o uso de automação no contexto do TDD [6].

A Figura 5.1 detalha as respostas de cada uma das perguntas para as ferramentas avaliadas.

Tool / Goal	AutoIt	JFCUnit	Selenium	FitNesse	Proven!
G1	difficult test first approach	difficult test first approach	easy test first approach	easy test first approach	easy test first approach
G2	non web application	non web application	web application	web application	web application
G3	development-like environment	development-like environment	client-like environment	client-like environment	client-like environment
G4	possible integration	possible integration	possible integration	possible integration	possible integration
G5	coding	Capture&Replay	Capture&Replay	coding	Capture&Replay
G6	id-based	id-based	id-based	-	id-based
G7	db interaction through integration with other tool	db interaction through integration with other tool	db interaction through integration with other tool	db interaction possible	db interaction possible
G8	error checking possible	error checking possible	error checking possible	error checking not possible	error checking possible
G9	non readable	non readable	hardly readable	readable	readable
G10	customer can not be involved	customer can not be involved	customer can not be involved	customer can be involved	customer can be involved

Figura 5.1: Resultado do estudo para avaliação das ferramentas de automação de testes (Fonte: [6]).

Capítulo 6

A Ferramenta Selenium

Neste capítulo, que possui o objetivo de abordar informações sobre a ferramenta de automação de testes Selenium, são descritas algumas características a respeito dessa ferramenta de testes. Sendo assim, inicialmente o Selenium será caracterizado, apresentando alguns dos seus aspectos históricos. Na sequência, são relacionadas as principais características do conjunto de ferramentas que compõem o Selenium, dando enfoque às principais funcionalidades de cada uma das ferramentas. Esse destaque será dado para o Selenium por se tratar de uma ferramenta consolidade e bastante completa para a automação de testes.

6.1 Definição e histórico

O Selenium pode ser definido como um conjunto de diferentes ferramentas de software, onde cada uma possui um propósito específico para auxiliar no processo de automação de testes [12]. Esse conjunto de ferramentas propicia uma grande coleção de funcionalidades de teste, baseada nas principais necessidades de teste das aplicações web, possibilidade uma grande flexibilidade na realização dos testes. O Selenium é uma ferramenta portátil de código aberto, que prevê suporte para diferentes navegadores web, aplicações web e tecnologias.

O projeto do Selenium foi iniciado em 2004 pelo desenvolvedor Jason Huggins, enquanto ele estava realizando testes para uma aplicação interna da empresa ThoughtWorks, uma consultoria global em tecnologia da informação com enfoque em desenvolvimento ágil de software. Sua idealização foi baseada na observação de que o seu tempo seria melhor aproveitado se ele não precisasse realizar testes manuais a cada alteração realizada no software, ainda mais se tratando do mesmo conjunto de testes que estava sendo realizado. Para solucionar esse problema, o desenvolvedor criou uma biblioteca em Javascript que realiza interações com a página web, possibilitando a execução de um conjunto de testes

múltiplas vezes de maneira automática em diversos navegadores web. Essa biblioteca eventualmente se tornou as primeiras ferramentas do Selenium, contendo o Selenium RC e o Selenium IDE [12].

Apesar do avanço significativo que a ferramenta representava, ele ainda apresentava algumas desvantagens que precisavam ser ajustadas. Como a automação dos testes era baseada em Javascript, algumas procedimentos eram complicados de ser realizados, tendo em vista que os navegadores web possuem restrições de segurança para o Javascript [12]. Além disso, a crescente evolução tecnológica tornou as aplicações web cada vez mais poderosas, dificultando ainda mais o trabalho de automação com os recursos até então existentes.

Com o propósito de resolver as principais dificuldades encontradas na automação de testes pelo Selenium, em 2006 um engenheiro do Google chamado Simon Stewart trabalhou em um projeto chamado de WebDriver. O seu objetivo era de criar uma ferramenta de testes que se comunicasse diretamente com o navegador web e sistema operacional, evitando restrições da tecnologia Javascript [12]. Alguns anos depois, em 2008, o Selenium e o WebDriver foram combinados, tornando o Selenium WebDriver como mais uma das ferramentas disponíveis.

6.2 Componentes do Selenium

Conforme informado anteriormente, o Selenium não é apenas uma única ferramenta, mas sim um conjunto de ferramentas para automação de navegadores web. As ferramentas que fazem parte do pacote Selenium são: Selenium IDE, Selenium RC, Selenium WebDriver e Selenium Grid. Essa variedade de ferramentas garante que o Selenium possa ser executado em diversos navegadores web e sistemas operacionais, além de também poder ser controlado por diferentes linguagens de programação e *frameworks* de teste [9].

O Selenium IDE é uma ferramenta de prototipação para desenvolvimento de *scripts* de teste. Ele possui plugins para os navegadores Firefox e Chrome, garantindo uma interface amigável e de fácil utilização para geração de testes automáticos [12]. Essa ferramenta possui uma funcionalidade de captura, onde as ações do usuário são gravadas, de modo que os seus passos são armazenados em um *script* reutilizável em uma das linguagens de programação suportadas, podendo ser executado novamente. Entretanto, essa ferramenta foi idealizada somente como uma ferramenta rápida de prototipação, não devendo ser usada para testes mais complexos e robustos. A Tabela 6.1 detalha algumas funcionalidades do Selenium IDE.

O Selenium RC foi umas das ferramentas precursoras do Selenium, sendo o seu principal projeto por um longo período de tempo. Entretanto, com a chegada do Selenium

Tabela 6.1: Funcionalidades do Selenium IDE (Fonte: [9]).

Número	**Funcionalidade**
1	Fácil gravação e re-execução
2	Seleção de campos inteligente, utilizando nome e identificadores
3	Todos os comandos comuns do Selenium completados automaticamente
4	Realiza testes práticos (testes de apresentação)
5	Pode realizar depuração com indicação de pontos de parada
6	Salva os testes no formato HTML, Ruby, Javascript, etc
7	Suporta arquivos com extensão .js
8	Possui opção para declarar automaticamente o nome de cada página
9	É customizável por meio de plugin

WebDriver, essa ferramenta foi descontinuada e não é mais amplamente utilizada [12]. Ela também é conhecida como Selenium 1.0 e permitia que testes automáticos fossem desenvolvidos para aplicações web. A Tabela 6.2 detalha as principais funcionalidades do Selenium RC.

Tabela 6.2: Funcionalidades do Selenium RC (Fonte: [9]).

Número	**Funcionalidade**
1	Foi o *framework* de teste carro-chefe do projeto do Selenium
2	Permite que o usuário utilize sua linguagem de programação de preferência
3	Suporta testes dirigidos a dados
4	Permite a execução de testes automáticos por quantas vezes o usuário quiser
5	Pode suportar novos navegadores web
6	Possui maturidade e API completa
7	Execução mais rápida que o Selenium IDE
8	Multi-navegadores e multi-plataforma

As limitações observadas do Selenium RC foram inspiração para a criação do Selenium WebDriver. Também chamado de Selenium 2, ele representa o conjunto de funcionalidades mais recente que foram adicionadas ao Selenium. Essa ferramenta possui uma interface de programação mais simples e concisa, ainda garantindo um suporte melhor para as páginas web dinâmicas [9]. A Tabela 6.3 detalha as principais funcionalidades do Selenium WebDriver.

Por fim, o Selenium Grid é uma solução que possibilita a escalabilidade do Selenium RC para execução de testes em larga escala e em múltiplos ambientes. Essa ferramenta permite que os testes sejam executados paralelamente em diferentes máquinas, distri-

Tabela 6.3: Funcionalidades do Selenium WebDriver (Fonte: [9]).

Número	**Funcionalidade**
1	É um *framework* de automação web que possibilita a execução de testes em diferentes navegadores web
2	Também possibilita o uso da linguagem de programação de preferência para criação dos *scripts* de teste
3	Sua arquitetura é mais simples que a arquitetura do Selenium RC
4	É mais rápido que o Selenium RC, pois executa o navegador pelo seu próprio motor de execução
5	É mais simples que o Selenium RC, não incluindo comandos desnecessários e complicados
6	Pode suportar o navegador web HtmlUnit sem cabeçalho

buindo a execução do teste. Com o a chegada do Selenium WebDriver, o Selenium Grid também teve o lançamento de uma nova versão para operar com a nova funcionalidade.

Portanto, considerando as características das ferramentas anteriormente elencadas, é possível destacar as seguintes funcionalidades para os componentes do Selenium: interface amigável para criação e execução de testes; robustez, flexibilidade e extensibilidade; capacidade de gerar casos de testes específicos e mais abrangentes; capacidade de gerar resultados da execução de testes de maneira detalhada, com consolidação e imagens de erro [9].

Parte II

Prática

Capítulo 7

Elementos dos processos adotados

Neste capítulo será apresentado o processo de avaliação da ferramenta de automação de testes durante o desenvolvimento de parte de uma aplicação web utilizando o TDD. Para isso, serão apresentadas as principais características da aplicação web desenvolvida (conteúdo, apresentação, navegação etc.), além do detalhamento dos processos e técnicas que serão adotados durante o estudo. Os modelos e processos apresentados nos capítulos anteriores foram utilizados para embasamento do projeto, servindo como referência para documentação dos componentes da aplicação web desenvolvida.

7.1 Características da aplicação web

A aplicação web desenvolvida no projeto é um jogo *online* onde os usuários devem escalar equipes virtuais com jogadores reais de um campeonato de futebol. Esse tipo de jogo é chamado de *Fantasy Game* e as equipes virtuais dos usuários competem de acordo com a pontuação dos atletas que foram escalados para o seu time, sendo que a pontuação é baseada em diversos indicadores de desempenho estatístico, como: número de gols marcados, número de faltas cometidas, números de finalizações, número de finalizações, entre outros.

O usuário pode escalar o seu time mediante a compra e venda de jogadores, que está condicionada ao valor financeiro que cada time possui. Todos os times iniciam com um valor padrão, mas esse valor é alterado a cada rodada, com base na valorização ou desvalorização dos jogadores escalados. Os pontos dos times são compilados a cada rodada e o time que obter a maior quantidade de pontos no final do campeonato é considerando o campeão.

Essa aplicação web foi batizada como CartolaUnB, fazendo referência ao *Fantasy Game* Cartola FC. O projeto da aplicação foi iniciado no contexto do projeto final da disciplina Banco de Dados (código 116378) na Universidade de Brasília (UnB), em um trabalho

realizado em grupo no semestre 1/2019, onde foram desenvolvidas as seguintes funcionalidades:

- Criação de novo usuário;
- Autenticação de usuário;
- Consulta de dados do usuário;
- Consulta de time do coração;
- Consulta de time do CartolaUnB;
- Cadastro de time do CartolaUnB;
- Alteração de dados do time do CartolaUnB;
- Exclusão de time do CartolaUnB;
- Consulta escalação atual do time do CartolaUnB;
- Consulta histórico de escalação do time do CartolaUnB;
- Consulta jogadores disponíveis para escalação;
- Consulta de dados de pontuação disponíveis;
- Consulta de campeonatos em andamento;
- Consulta classificação do campeonato.

A Figura 7.1 apresenta a visão da página principal do jogo CartolaUnB.

Figura 7.1: Página principal do CartolaUnB.

As funcionalidades foram desenvolvidas com base em uma visão de jogador, sendo necessário ainda existir uma visão de administrador do sistema para gerenciamento da

aplicação e inclusão dos indicadores de desempenho dos jogadores. Entretanto, essa visão de administrador do sistema não foi implementada. Portanto, em um primeiro momento, a administração da funcionalidade, com a inclusão do desempenho estatístico dos jogadores, precisa ser feita por meio do banco de dados.

Para o contexto do projeto descrito neste documento, foram definidas três novas funcionalidades a serem implementadas no sistema CartolaUnB utilizando o TDD, conforme itens descritos a seguir:

- Alteração de dados do usuário;
- Consulta formações táticas disponíveis;
- Consulta premiação do campeonato.

As funcionalidades do CartolaUnB foram definidas e o seu funcionamento especificado com base nos jogos já existentes nesse mesmo formato, como o Cartola FC.

7.2 Levantamento de requisitos

As novas funcionalidades do CartolaUnB foram desenvolvidas utilizando o TDD. Sendo assim, o processo de desenvolvimento foi baseado nos passos indicados por Turhan et al. [26], que iniciam com a seleção de uma pequena tarefa, escrita do teste para essa tarefa, execução dos testes para verificar se estão falhando, escrita do mínimo código para conclusão da tarefa, nova execução dos testes para verificar se estão passando, refatoração do código (se necessário) e repetição do processo a partir do primeiro passo.

As tarefas desenvolvidas foram determinadas a partir das três funcionalidades escolhidas para implementação no contexto do projeto. Essas tarefas foram descritas no formato de histórias do usuário, como sugerido no método de desenvolvimento ágil XP. A descrição das funcionalidades por meio de histórias de usuário possibilita a representação da necessidade do usuário de maneira simples, sem a exigência da elaboração de uma documentação extensa e burocrática. A documentação dos requisitos nesse formato garantiu maior liberdade ao processo de implementação do código, considerando que as histórias do usuário explicitam apenas as questões mais relevantes para o desenvolvimento.

Geralmente, as histórias de usuário são escritas à mão em papel. Porém, no contexto do projeto, optou-se pela representação digital das histórias de usuário em formato de itens, com a descrição da história como item principal e, como seus subitens, a relação das formas de utilização da funcionalidade. Os itens abaixo apresentam as histórias de usuário elaboradas para as três funcionalidades implementadas no projeto.

1. Eu, como usuário do sistema, gostaria de efetuar alterações nos dados do meu usuário para atualizar as informações de nome, CPF, data de nascimento, sexo, e-mail e senha:

    - O sistema deve apresentar a tela de alteração dos dados do usuário, com seis campos para preenchimento de texto e um botão para salvar as alterações;
    - O sistema deve informar ao usuário que as alterações foram realizadas com sucesso;
    - O sistema deve salvar os dados alterados no banco de dados.

2. Eu, como usuário do sistema, gostaria de consultar as formações táticas disponíveis no jogo:

    - O sistema deve recuperar os dados cadastrados no banco de dados com as formações táticas disponíveis.

3. Eu, como usuário do sistema, gostaria de consultar as premiações do campeonato:

    - O sistema deve recuperar os dados cadastrados no banco de dados com as premiações definidas para o campeonato.

Essas histórias de usuário foram desenvolvidas a partir da experiência vivenciada em outros jogos existentes no mesmo formato. Com base nelas, os testes foram então desenvolvidos e automatizados, para que somente depois as funcionalidades fossem implementadas. Esse processo de desenvolvimento representa o ciclo de aceitação do TDD [5], sendo realizado de maneira iterativa até a conclusão de todas as histórias de usuário.

Os testes foram desenvolvidos por meio do Selenium e, após a implementação dessas funcionalidades, a ferramenta de testes foi então avaliada com base nos critérios especificados.

7.3 Projeto do banco de dados

Antes da implementação do CartolaUnB, inicialmente foi elaborado o modelo relacional para a aplicação. O modelo relacional é um modelo de dados que permite a representação gráfica dos elementos que compõem um banco de dados, descrevendo cada entidade no formato de uma tabela e evidenciando o seu relacionamento com os demais objetos por meio de ligações.

A criação do modelo relacional para o CartolaUnB auxiliou no projeto da aplicação, deixando claro o modelo em que as informações seriam armazenadas e poderiam ser consultadas. O modelo relacional do CartolaUnB foi criado no início do projeto no contexto

da disciplina Banco de Dados por meio da ferramenta MySQL Workbench, sendo estendido com a criação de novos elementos para esse projeto, conforme destacado em vermelho na Figura 7.2.

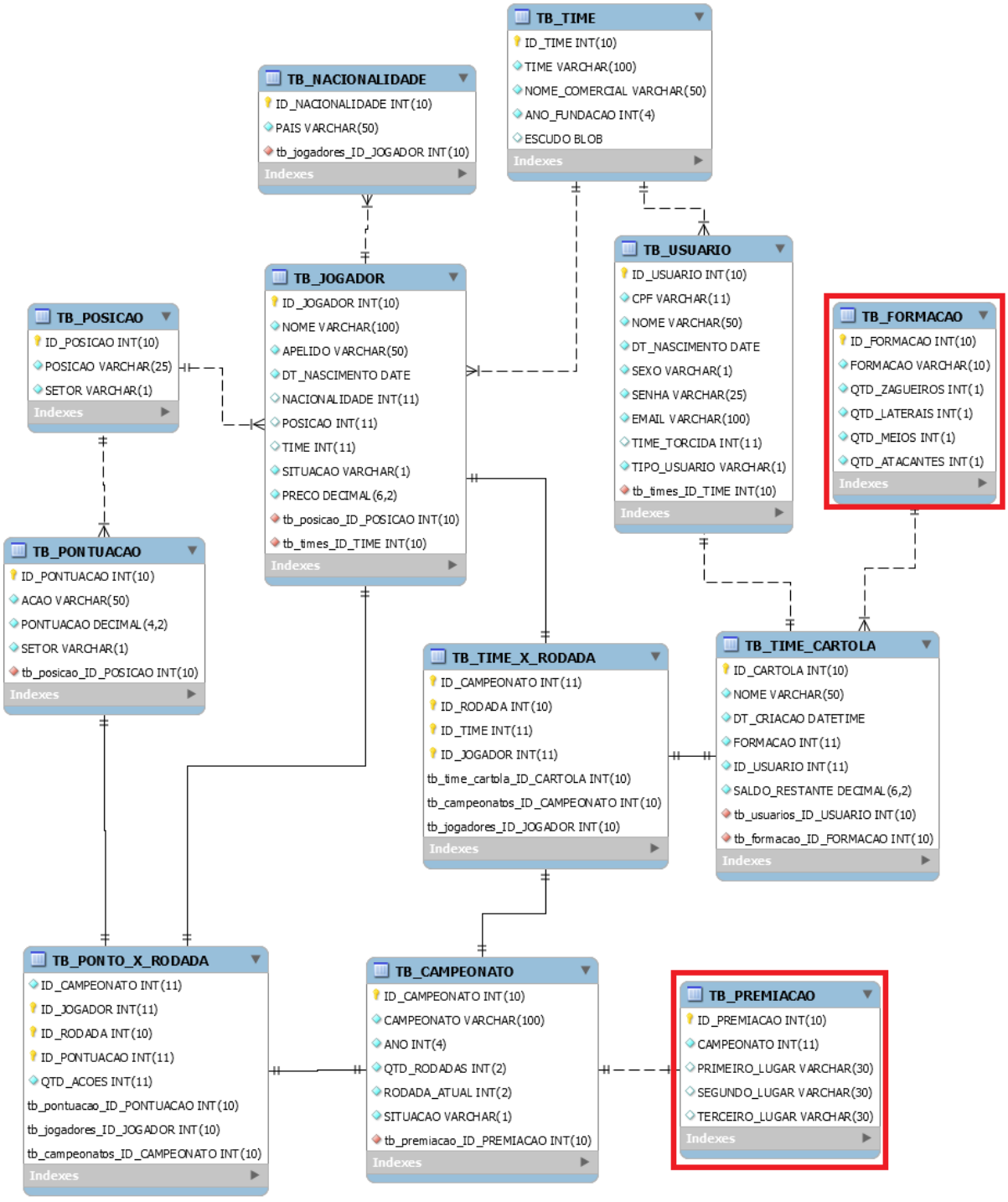

Figura 7.2: Modelo relacional do CartolaUnB.

Para o banco de dados relacional, a chave primária (PK) é um conjunto de campos

cujos valores não se repetem na tabela, podendo ser composta por um ou mais campos. Sendo assim, como o seu valor não pode se repetir ou ser nulo, a chave primária pode ser utilizada como um índice de referência para realizar para realizar consultas, incluir dados ou criar relacionamentos com outras tabelas do banco de dados.

O relacionamento entre as tabelas do banco de dados é criado por meio de uma chave estrangeira (FK). A chave estrangeira de uma tabela é um conjunto de campos de uma tabela que aponta para um conjunto de campos de outra tabela, fazendo a ligação entre as duas. Esse relacionamento é feito por meio da chave estrangeira para garantir a integridade dos dados, impossibilitando referenciar registros inexistentes ou excluir registros que possuem ligação.

Nessa representação do modelo relacional, as chaves primárias são representadas por uma pequena chave amarela ao lado do nome do campo e os relacionamentos entre as tabelas são representados por linhas que conectam as tabelas. Além disso, as chaves estrangeiras relacionadas com a tabela também são representadas após o nome dos campos da tabela, sendo identificadas por um losango vermelho ao seu lado.

As tabelas "TB_USUARIO", "TB_FORMACAO" e "TB_PREMIACAO" foram as principais consideradas no contexto do projeto, considerando que os registros precisariam ser acessados ou alterados nessas estruturas para as três funcionalidades implementadas nesse contexto.

7.4 Elementos da arquitetura do software

Conforme descrito na primeira seção deste capítulo, a aplicação web já possuía algumas funcionalidades implementadas. No contexto desse projeto, foram implementadas novas funcionalidades para a aplicação. A arquitetura do software já havia sido definida anteriormente, sendo seguida durante esse projeto.

A arquitetura do software foi definida de modo a haver uma segregação entre os elementos de apresentação do software (telas) e a persistência dos dados (acesso ao banco de dados), criando diferentes camadas para a aplicação. Essa arquitetura está de acordo com o padrão *Data Access Object* (DAO), que possibilita alterações na forma de persistência dos dados sem mudanças na interface do usuário.

Esse padrão é recomendado para que o modo de busca e gravação dos dados seja abstraído, deixando esse aspecto transparente para a aplicação e facilitando o desenvolvimento e manutenção do código. O padrão também está de acordo com o estilo de arquitetura Modelo Visão Controle (MVC).

Para possibilitar essa segregação no projeto, foram criados arquivos com o código da interface com o usuário em HTML e PHP sem nenhuma interação com o banco de

dados. Quando necessário, esses códigos fizeram referência à funções criadas em PHP especificamente para realizar autenticação, consultas, exclusão e atualização de dados por meio de chamadas para o sistema gerenciador do banco de dados. Essas funções foram criadas em arquivos diferentes, que foram disponibilizados em pastas diferentes no projeto, caso fosse necessário implementar restrições de acesso para algum usuário.

Os detalhes dessa estrutura de arquitetura são exemplificados no próximo capítulo, durante a implementação das funcionalidades.

7.5 Projeto dos casos de teste

Considerando que o processo de desenvolvimento utilizado foi guiado por teste (TDD), também foi necessário elaborar os casos de teste antes de iniciar a implementação do código. Como as funcionalidades a serem testadas ainda não existiam, os casos de teste foram elaborados de maneira simples, considerando o comportamento da aplicação em um cenário em que o teste seria concluído com sucesso.

Cada uma das três funcionalidades selecionadas para o desenvolvimento no projeto tiveram um caso de teste projetado. Esses casos de teste foram construídos considerando o caminho que o usuário precisaria seguir na aplicação para acessar a funcionalidade requerida. Em todos os casos de teste, considerou-se que a aplicação estaria sendo acessada a partir da página de *login* da aplicação, sendo necessário primeiramente informar os dados de acesso para somente depois acessar a funcionalidade.

O comportamento das funcionalidades da aplicação web foi considerado a partir dos seus requisitos, descritos anteriormente neste capítulo, para criação dos casos de teste.

A Tabela 7.1 apresenta o caso de teste para a funcionalidade de consulta das formações táticas disponíveis no CartolaUnB.

Tabela 7.1: Caso de teste para consulta de formações táticas disponíveis.

Item de teste	Consulta formações táticas disponíveis
Localização	Tela de *login* da aplicação (localhost/cartola/aplicacao/)
Especificações de entrada	1. CPF: 11111111111; 2. Senha: 123.
Procedimentos	1. Inserir o dado "CPF"na área "CPF", o dado "Senha"na área "SENHA"e clicar no botão "Acessar"; 2. Clicar na opção "Informações do Cartola"no menu do canto esquerdo da tela para expandir as funcionalidades e clicar na opção "Formações Disponíveis".
Especificações de saída	Após os procedimentos, a tela deve ser alterada para apresentar uma nova tela com as formações táticas cadastradas para o jogo

A Tabela 7.2 apresenta o caso de teste para a funcionalidade de consulta da premiação do campeonato no CartolaUnB.

Tabela 7.2: Caso de teste para consulta de premiação do campeonato.

Item de teste	Consulta premiação do campeonato
Localização	Tela de *login* da aplicação (localhost/cartola/aplicacao/)
Especificações de entrada	1. CPF: 11111111111; 2. Senha: 123.
Procedimentos	1. Inserir o dado "CPF"na área "CPF", o dado "Senha"na área "SENHA"e clicar no botão "Acessar"; 2. Clicar na opção "Informações do Cartola"no menu do canto esquerdo da tela para expandir as funcionalidades e clicar na opção "Premiação".
Especificações de saída	Após os procedimentos, a tela deve ser alterada para apresentar uma nova tela com a premiação cadastrada para o campeonato

A Tabela 7.3 apresenta o caso de teste para a funcionalidade de alteração de dados do usuário no CartolaUnB.

Tabela 7.3: Caso de teste para alteração de dados do usuário.

Item de teste	Alteração de dados do usuário
Localização	Tela de *login* da aplicação (localhost/cartola/aplicacao/)
Especificações de entrada	1. CPF: 11111111111; 2. Senha: 123; 3. Nome: Rafael Santori; 4. DtNasc: 2000/01/01; 5. Sexo: M; 6. Email: teste@teste.com.br.
Procedimentos	1. Inserir o dado "CPF"na área "CPF", o dado "Senha"na área "SENHA"e clicar no botão "Acessar"; 2. Clicar no botão "Editar"presente na tela inicial da aplicação; 3. Inserir o dado "CPF"na área "CPF, o dado "Nome"na área "Nome", o dados "DtNasc"na área "Data de Nascimento", o dado "Sexo"na área "Sexo", o dado "Email"na área "Email", o dado "Senha"na área "Senha"e clicar no botão "Atualizar".
Especificações de saída	Após os procedimentos, os novos dados devem ser gravados no banco de dados e a tela deve ser alterada para apresentar a tela inicial da aplicação

No que se refere à classe dos casos de teste, é possível considerá-los como testes de caixa preta, considerando que eles foram baseados somente nas especificações e ações que a aplicação deveria desempenhar. O foco dessa classe de teste está nos requisitos do sistema, buscando verificar o seu comportamento a partir de entradas aceitas e saídas esperadas. Não foi possível realizar testes de caixa branca, tendo em vista que esses testes

são baseados na estrutura do código fonte e, durante a criação dos casos de teste, o código fonte ainda não havia sido desenvolvido.

Com relação à fase de aplicação do caso de teste, como o objetivo principal do teste era a verificação isolada da nova funcionalidade que seria desenvolvida, cada caso de teste considerou apenas um módulo isolado da aplicação web. Um teste foi realizado para cada caso de teste procurando garantir o funcionamento correto da funcionalidade.

A partir da definição dos casos de teste, a implementação da aplicação web pôde ser iniciada, partindo do desenvolvimento dos *scripts* de teste no Selenium para depois desenvolver as funcionalidades por meio do TDD.

7.6 Critérios para avaliação do Selenium

Após o uso do Selenium para execução dos testes realizados durante o processo de desenvolvimento da aplicação web, foi possível realizar uma avaliação da ferramenta com base em alguns critérios e métricas estabelecidos inicialmente. Nesse contexto, o processo de avaliação da ferramenta foi baseado no trabalho de Jureczko e Mlynarski [6], sendo realizado em três fases principais, conforme itens a seguir:

1. Criação dos casos de teste;
2. Implementação do código;
3. Verificação das métricas e critérios de avaliação da ferramenta.

Os critérios para avaliação da ferramenta foram definidos com base nos dados apresentados no Capítulo 5. Como os testes da ferramenta foram realizados em um cenário bem específico e restrito, os critérios relacionados ao fornecedor, custo e benefício, hardware e software e opiniões de fontes técnicas e não técnicas não foram considerados para a avaliação. Sendo assim, os critérios considerados para a avaliação estão relacionados basicamente com as características funcionais e de qualidade da ferramenta.

As questões foram elaboradas com base em critérios de avaliação identificados na literatura, de maneira semelhante ao trabalho de Jureczko e Mlynarski [6], conforme descrito no Capítulo 5. O trabalho dos autores foi utilizado como referência para a elaboração de cinco perguntas. No entanto, diferente do trabalho em questão, nesse projeto nenhuma opção de resposta foi pré-estabelecida.

As questões buscam descrever as características e funcionalidades suportadas pela ferramenta, conforme itens abaixo:

- A ferramenta de automação de testes propicia a utilização do TDD?

- A ferramenta de automação é de simples aprendizado e utilização por parte dos usuários?
- Quais são os recursos que a ferramenta possui para criação dos casos de teste?
- Os casos de testes criados por meio da ferramenta são de fácil compreensão por pessoas que não são profissionais da área?
- A ferramenta de automação utiliza identificadores ou posições baseadas em coordenadas para mapear os elementos da aplicação web?
- A ferramenta de automação possui integração com banco de dados?
- A ferramenta de automação pode ser integrada com outros recursos e tecnologias?

Portanto, após o desenvolvimento da aplicação e execução dos testes, as questões foram respondidas com base na percepção individual do desenvolvedor e pesquisas realizadas sobre o Selenium.

Capítulo 8

Execução do processo de desenvolvimento

Após o planejamento e especificação do projeto, iniciou-se o desenvolvimento da aplicação web utilizando as técnicas e modelos explicitados no capítulo anterior. Inicialmente, os recursos tecnológicos a serem utilizados durante a implementação foram definidos a partir de uma pesquisa das tecnologias mais adequadas ao contexto do projeto.

Neste capítulo serão apresentados os elementos do processo de desenvolvimento da aplicação web, com destaque para os testes realizados durante o processo. Para isso, inicialmente será apresentada uma breve descrição a respeito das tecnologias utilizadas. Posteriormente, serão apresentadas informações sobre os elementos de implementação dos casos de teste criados e demais elementos presentes no desenvolvimento das novas funcionalidades da aplicação web.

8.1 Elementos da implementação

Para o desenvolvimento da aplicação web, foi necessário selecionar uma linguagem de programação consolidada e amplamente utilizada. Essa característica facilitou a busca por fontes de dados para auxiliar no processo de desenvolvimento, além de também minimizar os problemas relacionados à integração com o banco de dados e outras tecnologias. A aplicação web foi desenvolvida em PHP, com o *frontend* desenvolvido em HTML, JavaScript e CSS, utilizando também o *framework* Bootstrap com o intuito de tornar a interface gráfica amigável. O desenvolvimento da aplicação foi realizado utilizando o editor de texto Sublime Text.

O sistema gerenciador de banco de dados (SGBD) utilizado para o desenvolvimento da aplicação foi o MySQL, sendo acessado por meio da ferramenta phpMyAdmin e também

pelo software MySQL Workbench. Esse SGBD foi selecionado devido à sua popularidade e também por possuir integração com a linguagem PHP.

Essas tecnologias foram acessadas por meio do XAMPP, um pacote com os principais servidores de código aberto do mercado, contendo o MySQL e Apache com suporte à linguagem PHP. O código fonte do projeto foi disponibilizado na pasta de instalação do XAMPP e, com a inicialização dos servidores do XAMPP, a aplicação pode então ser acessada por meio do endereço http://localhost/cartola/aplicacao/ em um navegador web. O navegador web utilizado para navegação na aplicação web durante o projeto foi o Google Chrome.

Por fim, o Selenium foi a ferramenta de testes utilizada para automação e execução dos testes da aplicação web. Os testes foram desenvolvidos com o Selenium WebDriver, utilizando o ambiente integrado de desenvolvimento (IDE) Microsoft Visual Studio para criação dos *scripts* de testes. Em alguns momentos, o Selenium IDE também foi usado para realização de testes da aplicação.

O projeto foi desenvolvido e executado em um computador com o sistema operacional Windows 10 Home, contendo um processador 8ª Geração Intel Core i7, com 8GB de memória RAM.

8.2 Elementos de implementação dos casos de teste

Como a implementação das novas funcionalidades foi realizada utilizando o TDD, o desenvolvimento foi iniciado por meio da criação dos *scripts* de automação de teste. Esses *scripts* foram construídos no Selenium a partir dos casos de teste definidos no Capítulo 7, sendo criado um *script* de teste para cada caso de teste, conforme representado na Figura 8.1.

Todos os *scripts* de teste possuíam uma parte em comum, responsável pelo acesso à URL da aplicação e *login* do usuário, como pode ser observado na Figura 8.2. Os demais componentes do *script* de teste eram específicos da funcionalidade que estava sendo testada.

Os *scripts* de teste foram construídos seguindo o mesmo processo e com estruturas bem semelhantes, onde o teste deveria clicar em botões e *links* específicos na tela para acessar determinadas funcionalidades. Somente a funcionalidade para alteração de dados do usuário possuía alguns elementos a mais, pois ela envolve também o preenchimento de campos para atualização das informações do usuário. Portanto, como essa funcionalidade possuía o cenário de teste mais completo, ela foi considerada para exemplificação ao longo deste capítulo.

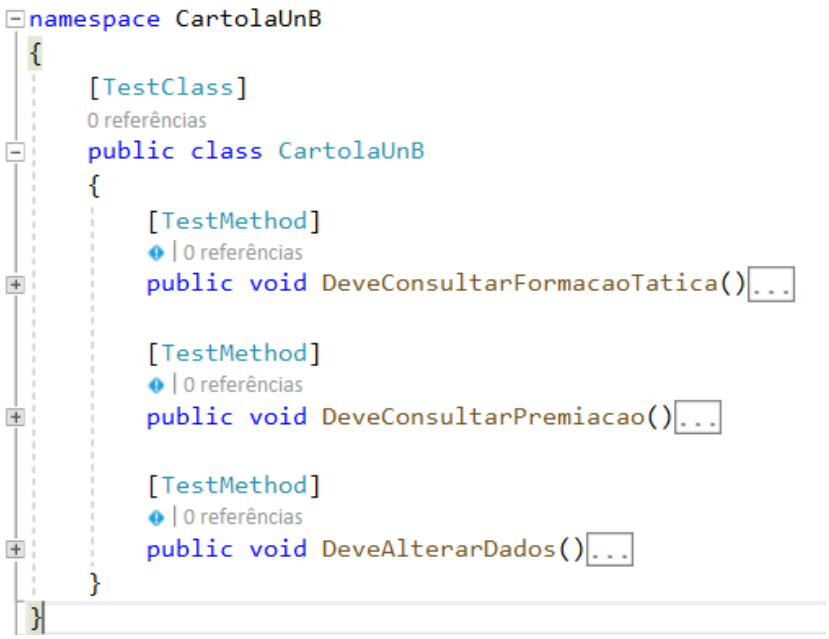

Figura 8.1: Estrutura do projeto de automação de teste.

```
// Inicializar o Chrome
using (var driver = new ChromeDriver())
{
    // Maximizar a página
    driver.Manage().Window.Maximize();

    // Acessar a página do Cartola UnB
    driver.Navigate().GoToUrl("http://localhost/cartola/aplicacao/");

    // Preencher o CPF
    var usuario = driver.FindElementById("cpf");
    usuario.SendKeys("11111111111");

    // Preencher a senha
    var senha = driver.FindElementById("senha");
    senha.SendKeys("123");

    // Clicar no botão Acessar
    driver.FindElementByName("Acessar").Click();
```

Figura 8.2: *Script* de teste para acesso e *login* na aplicação.

O *script* de teste da funcionalidade para alteração de dados do usuário foi desenvolvido com base na Tabela 7.3 e pode ser visto na Figura 8.3.

```
// Preencher as informações para edição
driver.FindElementById("cpf").SendKeys("11111111111");
driver.FindElementById("nome").SendKeys("Rafael Santori");
driver.FindElementById("dt_nascimento").SendKeys("2000/01/01");
driver.FindElementById("sexo").SendKeys("M");
driver.FindElementById("email").SendKeys("teste@teste.com.br");
driver.FindElementById("senha").SendKeys("123");

// Clicar no botão Atualizar
driver.FindElementByName("atualizar").Click();
```

Figura 8.3: *Script* de teste para alteração de dados do usuário.

8.3 Elementos de implementação do software

Conforme descrito anteriormente no Capítulo 7, a aplicação web já possuía algumas funcionalidades implementadas. No contexto desse projeto, foram implementadas somente novas funcionalidades para a aplicação. Sendo assim, a aplicação já estava configurada para conexão com o sistema gerenciador de banco de dados MySQL. A conexão estava sendo realizada conforme apresentado na Figura 8.4, utilizando a biblioteca *PHP Data Objects* (PDO).

```php
<?php
// página responsável pela conexão com o banco de dados MySQL

class Database{
    //utiliza os valores abaixo como parâmetros de conexão
    private $dsn = "mysql:host=localhost;dbname=db_cartola";
    private $username = "root";
    private $password = "";
    public $conn;

    //função responsável por realizar a conexão com o banco de dados
    public function getConnection(){
        $this->conn = null;
        //conexão realizada com a extensão PDO
        try {
            $this->conn = new PDO($this->dsn, $this->username, $this->password);
            $this->conn->exec("set names utf8");
        }
        catch(PDOException $exception) {
            echo "Connection error: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
?>
```

Figura 8.4: Código da classe para realizar a conexão com o banco de dados.

Após a criação dos *scripts* de teste detalhados na seção anterior, o desenvolvimento de cada funcionalidade foi iniciado. Esta seção apresenta detalhes do processo de desenvolvimento da funcionalidade de alteração dos dados do usuário.

Inicialmente, o teste foi executado sem que a funcionalidade tivesse sido implementada. Esse fluxo está de acordo com o TDD, onde o processo de desenvolvimento tem início com a falha do teste. A Figura 8.5 mostra o teste sendo executado com erro no início da implementação, pois os elementos não são identificados na página.

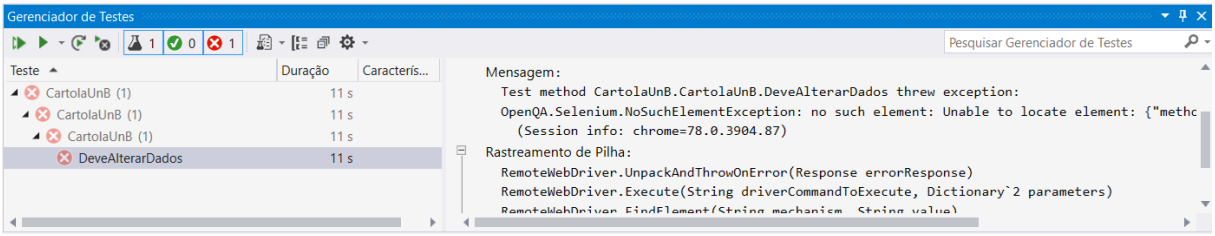

Figura 8.5: Execução de teste com erro para alteração de dados do usuário.

Após a execução do teste com erro, a codificação foi iniciada para implementar a nova funcionalidade considerada. Durante o desenvolvimento, o *script* de teste foi executado repetidas vezes para verificar se a funcionalidade apresentava o comportamento adequado. A Figura 8.6 apresenta um trecho do código para criação da tela de alteração de dados do usuário.

```
<div class="box box-primary">
  <div class="box-header with-border">
    <h3 class="box-title">ATUALIZAR CADASTRO</h3>
  </div>
  <!-- /.box-header -->
  <!-- form start -->
  <form role="form">
    <div class="box-body">
      <div class="form-group">
        <label for="exampleInputName1">CPF</label>
        <input type="text" class="form-control" id="cpf" placeholder="Informe o CPF">
      </div>
      <div class="form-group">
        <label for="exampleInputName1">Nome</label>
        <input type="text" class="form-control" id="nome" placeholder="Informe o Nome">
      </div>
      <div class="form-group">
        <label for="exampleInputName1">Data de Nascimento</label>
        <input type="text" class="form-control" id="dt_nascimento" placeholder="Informe a data de
          nascimento">
      </div>
      <div class="form-group">
        <label for="exampleInputName1">Sexo</label>
        <input type="text" class="form-control" id="sexo" placeholder="Informe o sexo">
      </div>
```

Figura 8.6: Trecho de código para criação da tela de alteração de dados do usuário.

Além da tela, também foi implementada uma função para realizar a alteração dos dados do usuário na tabela do banco de dados. Essa função possui a chamada de um comando "UPDATE" para o sistema gerenciador do banco de dados, que efetua a atualização dos dados, como apresentado na Figura 8.7.

```php
// função responsável pela atualização dos dados do usuário
function update(){
    // query
    $query = "UPDATE ".$this->table_name." SET cpf='".$this->cpf."', email='".$this->email."', senha='".$this->senha."',
        nome='".$this->nome."', sexo='".$this->sexo."', dt_nascimento='".$this->dt_nascimento."' WHERE id_usuario=".$this->
        id_usuario;
    // prepara a query
    $stmt = $this->conn->prepare($query);
    // executa a query
    if($stmt->execute()){
        return true;
    }
    return false;
}
```

Figura 8.7: Código da função para atualização dos dados do usuário.

Essa função é chamada após a conexão com o banco de dados e a inicialização da sessão em um *script* na linguagem PHP, representado na Figura 8.8.

```php
<?php
 // incluir a conexão do banco de dados e os dados do objeto a ser consultado
include_once '../config/database.php';
include_once '../objects/tb_usuario.php';
// instância do banco de dados e objeto e inicia a sessão
$database = new Database();
$db = $database->getConnection();
session_start();
// inicialização do objeto
$usuario = new Usuario($db);
// carrega os valores das propriedades
$usuario->id_usuario = $_SESSION["id_usuario"];
$usuario->cpf = $_POST['cpf'];
$usuario->nome = $_POST['nome'];
$usuario->dt_nascimento = $_POST['dt_nascimento'];
$usuario->sexo = $_POST['sexo'];
$usuario->email = $_POST['email'];
$usuario->senha = $_POST['senha'];
// executa a consulta
if($usuario->update()){
    $usuario_arr=array(
        "status" => true,
        "message" => "Atualização realizada com sucesso!"
    );
}
else{
    $usuario_arr=array(
        "status" => false,
        "message" => "Erro nos dados informados!"
    );
}
print_r(json_encode($usuario_arr));
?>
```

Figura 8.8: Código com a chamada da função para atualização dos dados do usuário.

A chamada do *script* em PHP foi incluída em uma função que é executada a partir do botão de atualização dos dados de usuário na nova tela criada. Essa chamada também inclui a passagem dos valores digitados pelo usuário em cada um dos campos para atualização. A Figura 8.9 mostra a função com a chamada para o *script* PHP.

```
<script>
// função responsável pela chamada da atualização dos dados do usuário
    function UpdateUser(){
        $.ajax(
        {
            type: "POST",
            url: '../api/tb_usuario/update.php',
            dataType: 'json',
            data: {
                cpf: $("#cpf").val(),
                nome: $("#nome").val(),
                dt_nascimento: $("#dt_nascimento").val(),
                sexo: $("#sexo").val(),
                email: $("#email").val(),
                senha: $("#senha").val()
            },
            error: function (result) {
                alert(result.responseText);
            },
            success: function (result) {
                if (result['status'] == true) {
                    alert("Registro atualizado com sucesso!");
                    window.location.href = '/cartola/aplicacao/main.php';
                }
                else {
                    alert(result['message']);
                }
            }
        });
    }
</script>
```

Figura 8.9: Código da função para chamada do *script* PHP.

As demais funcionalidades passaram pelo mesmo processo de desenvolvimento, diferindo apenas nos artefatos gerados para cada uma delas. Como as outras funcionalidades realizam consultas à dados, elas possuem chamadas de comandos "SELECT" para o sistema gerenciador de banco de dados efetuar a busca dos registros. Além disso, a disposição dos elementos na tela é diferente, tendo em vista que não foi necessário criar campos para interação com o usuário, somente uma tabela para exibição dos resultados da consulta.

O desenvolvimento da funcionalidade foi concluído quando o teste não apresentou mais erros. Portanto, quando o *script* de teste foi executado com sucesso, o código foi refatorado para remover os elementos desnecessários e melhorar a disposição dos objetos desenvolvidos.

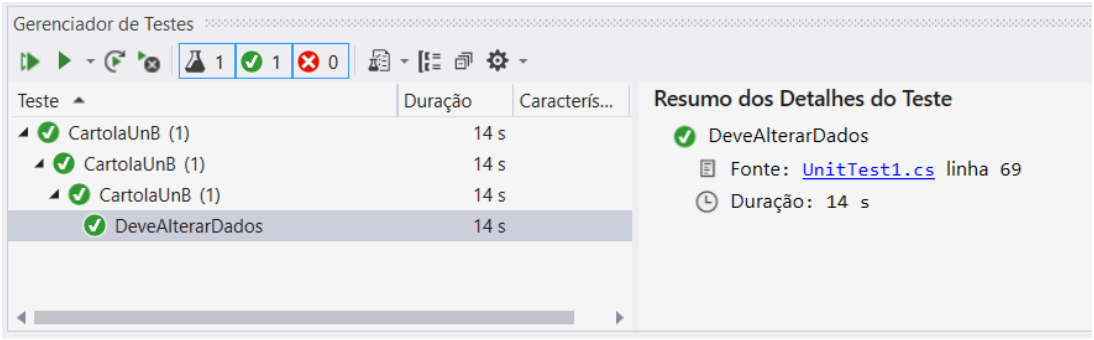

Figura 8.10: Execução de teste para alteração de dados do usuário com sucesso.

8.4 Elementos da interface com o usuário

A execução bem sucedida dos testes das funcionalidades indica que os requisitos funcionais estavam sendo cumpridos, ou seja, esse módulo da aplicação estava funcionando corretamente. Como não houve nenhuma especificação sobre os elementos da interface com o usuário, ela foi desenhada buscando apresentar os dados na tela com simplicidade. Além disso, buscou-se também manter a padronização da aplicação, considerando que ela já possuía outras funcionalidades. Portanto, o mesmo *layout* da interface foi utilizado para as três funcionalidades.

As subseções abaixo apresentam mais detalhes sobre o desenho da interface com o usuário para as três novas funcionalidades da aplicação web, explicando o funcionamento de cada uma.

8.4.1 Alteração de dados do usuário

A funcionalidade de alteração de dados do usuário possibilita a mudança dos principais dados do usuário, como: CPF, nome, data de nascimento, sexo, e-mail e senha. Essa funcionalidade pode ser acessada por meio do *link* "Editar" existente na página principal da aplicação. Com a atualização dos dados, uma mensagem é apresentada para o usuário indicando que a alteração foi realizada com sucesso e os dados são gravados no banco de dados.

A Figura 8.11 apresenta a funcionalidade para alteração de dados do usuário no CartolaUnB.

8.4.2 Consulta de formações táticas

A funcionalidade de consulta às formações táticas disponíveis pode ser acessada por meio do menu "Informações do Cartola", selecionando a opção "Formações Disponíveis". Ao

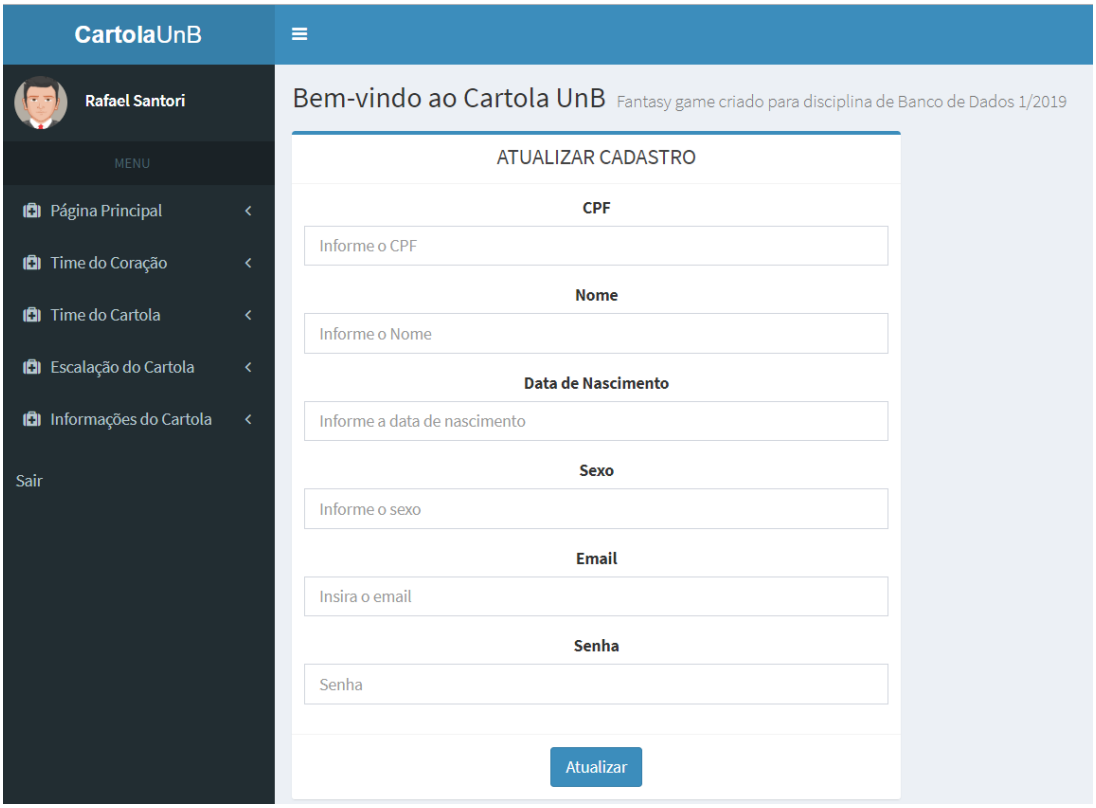

Figura 8.11: Funcionalidade para alteração de dados do usuário.

acessar a página, a aplicação apresenta uma lista com as formações táticas que estão cadastradas no banco de dados para o jogo.

A Figura 8.12 apresenta a funcionalidade para consulta às formações táticas disponíveis no CartolaUnB.

8.4.3 Consulta de premiação do campeonato

A funcionalidade de consulta à premiação do campeonato pode ser acessada por meio do menu "Informações do Cartola", selecionando a opção "Premiação". Ao acessar a página, a aplicação apresenta uma lista com premiacação cadastrada no banco de dados para o jogo.

A Figura 8.13 apresenta a funcionalidade para consulta à premiação do campeonato no CartolaUnB.

Figura 8.12: Funcionalidade para consulta às formações táticas disponíveis no jogo.

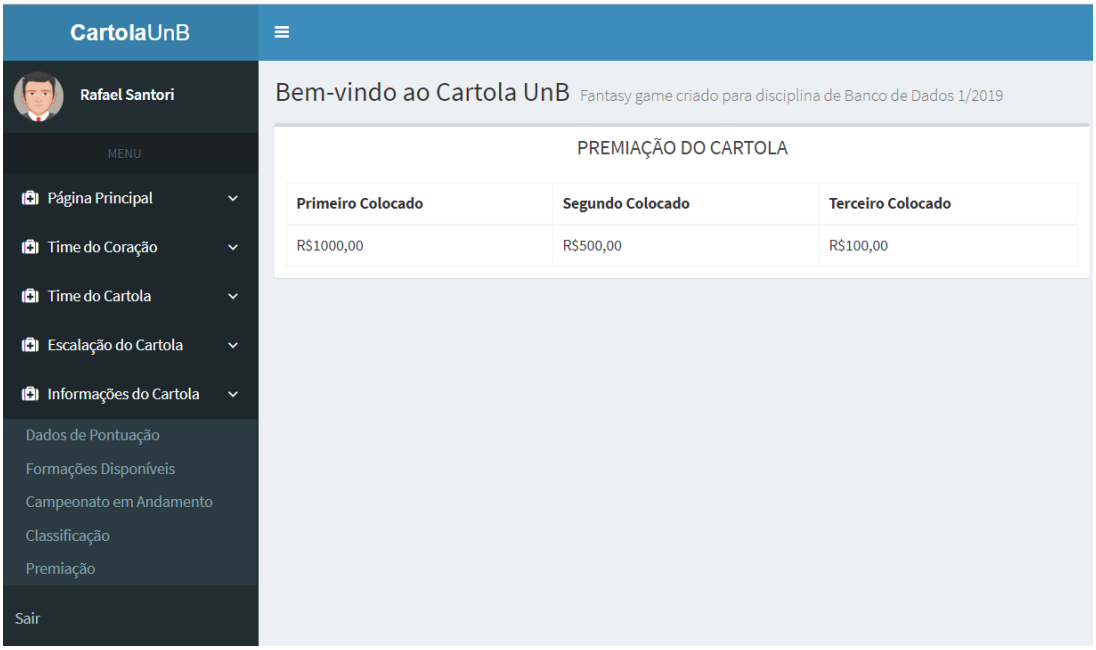

Figura 8.13: Funcionalidade para consulta à premiação do campeonato.

Capítulo 9

Elementos do processo de avaliação

Neste capítulo será apresentada a análise realizada durante o processo de avaliação do Selenium, ferramenta de automação de testes utilizada no projeto. Para isso, inicialmente uma breve introdução sobre a ferramenta é apresentada, descrevendo algumas das suas características. Em seguida, as questões definidas no Capítulo 7 a partir de alguns critérios de avaliação identificados na literatura são respondidas, descrevendo algumas funcionalidades da ferramenta de automação de testes.

9.1 Introdução

Conforme apresentado no Capítulo 6, o Selenium é um conjunto de ferramentas de software para automação de testes. Como não se trata apenas de uma ferramenta, inicialmente é necessário definir qual componente do Selenium que melhor se enquadra ao cenário do seu projeto de automação. Para o contexto deste trabalho, as ferramentas analisadas foram o Selenium IDE e o Selenium WebDriver. O Selenium Grid não foi utilizado, tendo em vista que o seu uso está associado a cenários de escalabilidade, assim como o Selenium RC também não foi usado, pois essa ferramenta foi descontinuada.

O Selenium IDE é uma ferramenta relevante para a criação rápida de *scripts* de teste, seja para realização de testes de exploração, qualidade ou regressão. Essa ferramenta funciona como um módulo de extensão do navegador web, também chamado de *plugin*, podendo ser instalado facilmente em diversos navegadores. A sua interface gráfica simples e a sua funcionalidade de criação de *scripts* de teste por meio de um módulo de captura possibilitam que a ferramenta seja utilizada por pessoas que não possuam conhecimentos de programação.

No início do projeto, o Selenium IDE foi instalado para verificar se o seu uso seria viável, já que a sua simplicidade iria favorecer a automação dos testes. Entretanto, como os *scripts* de teste deveriam ser desenvolvidos em um contexto do TDD e, consequentemente,

o *script* de teste seria desenvolvido antes da aplicação web, não seria possível utilizar a funcionalidade de captura para criação dos testes. Por isso, essa ferramenta não foi utilizada durante o desenvolvimento com o TDD, somente para criação de alguns *scripts* de teste para verificação de algumas funcionalidades que já estavam desenvolvidas da aplicação web. De qualquer forma, é válido destacar que esse é um dos principais recursos que o Selenium possui atualmente e, por isso, essa ferramenta também foi explorada no projeto.

O Selenium WebDriver é uma ferramenta mais completa, permitindo a criação de *scripts* de teste mais completos e robustos. Para que isso seja possível, os *scripts* de teste precisam ser criados na forma de códigos, não sendo compreensíveis de maneira simples por pessoas que não sejam da área ou que não possuam conhecimentos básicos de programação. A sua instalação também é mais complicada, tendo em vista que ele funciona como um *framework* que pode ser utilizado em diferentes ambientes integrados de desenvolvimento (IDE). Além disso, também é necessário instalar *drivers* do navegador web e da linguagem de programação que deseja-se utilizar. No contexto deste projeto, o Selenium WebDriver foi utilizado com a IDE Microsoft Visual Studio e com o navegador Google Chrome.

Após o processo de instalação do Selenium WebDriver e dos *drivers* necessários para que ele fosse executado, o processo de desenvolvimento dos *scripts* de teste pôde então ser iniciado. Considerando a falta de experiência do desenvolvedor com o Selenium, a documentação da ferramenta disponível em seu *site* [12] foi utilizada como referência para automação dos testes.

9.2 Elementos de avaliação do Selenium

Durante o desenvolvimento dos *scripts* de teste no Selenium WebDriver, foi possível trabalhar com diversos recursos distintos que a ferramenta possui. Outras funcionalidades foram melhor exploradas somente quando os testes estavam sendo executados ao longo do processo de desenvolvimento da aplicação web com o uso do TDD.

Os recursos e características observados da ferramenta serviram como insumo para que o processo de avaliação fosse realizado, sendo documentados na forma de respostas às questões previamente estabelecidas como critérios de avaliação no Capítulo 7, conforme itens abaixo:

1. A ferramenta de automação de testes propicia a utilização do TDD?
2. A ferramenta de automação é de simples aprendizado e utilização por parte dos usuários?

3. Quais são os recursos que a ferramenta possui para criação dos casos de teste?
4. Os casos de testes criados por meio da ferramenta são de fácil compreensão por pessoas que não são profissionais da área?
5. A ferramenta de automação utiliza identificadores ou posições baseadas em coordenadas para mapear os elementos da aplicação web?
6. A ferramenta de automação possui integração com banco de dados?
7. A ferramenta de automação pode ser integrada com outros recursos e tecnologias?

Portanto, as respostas para as questões e, consequentemente, para a avaliação do Selenium foram embasadas na experiência vivenciada durante o desenvolvimento e execução dos *scripts* de teste para o projeto. Como os *scripts* de teste focaram em cenários específicos e simplificados, não houve aprofundamento em alguns aspectos da ferramenta. Nessas situações, as respostas também foram embasadas em outras referências sobre a ferramenta, como principalmente a sua documentação.

É importante destacar que não foram pré-estabelecidas respostas para as perguntas, possibilitando que as questões fossem respondidas de maneira subjetiva e com mais detalhes. Essa estratégia foi usada procurando obter mais insumos para a avaliação.

As subseções abaixo contém as respostas de cada uma das perguntas com as suas respectivas justificativas.

9.2.1 Avaliação do Selenium quanto à utilização do TDD

Essa seção busca responder a questão: "A ferramenta de automação de testes propicia a utilização do TDD?".

Conforme destacado anteriormente no Capítulo 3, o TDD é uma abordagem em que os casos de teste são escritos antes do desenvolvimento da aplicação, considerando que a prática de desenvolvimento é guiada pelos testes. Nesse caso, para que seja possível utilizar a ferramenta de automação de testes em conjunto com o TDD, é necessário que a ferramenta possibilite o desenvolvimento dos testes mesmo sem que o produto de software ainda exista.

O Selenium IDE, com a criação dos seus testes baseada na captura de ações realizadas pelo usuário na aplicação, não possibilita uma abordagem guiada por testes. Apesar de ser o módulo mais simples para desenvolvimento dos casos de teste, a interação do usuário com a aplicação para captura das suas ações pressupõe que a aplicação já esteja pronta. Por isso, não é possível utilizar essa ferramenta com o TDD.

Para o Selenium WebDriver, a utilização do TDD é possível, tendo em vista que os casos de teste são escritos na forma de *scripts*. Nesse caso, não é preciso que a aplicação

esteja pronta para iniciar o desenvolvimento dos *scripts* de teste, sendo que a execução dos testes apresentará erro caso a aplicação ainda não exista ou esteja pronta, como deve ocorrer no início do desenvolvimento com a prática do TDD.

9.2.2 Avaliação do Selenium quanto à sua facilidade no aprendizado e utilização

Essa seção busca responder a questão: "A ferramenta de automação é de simples aprendizado e utilização por parte dos usuários?". Para analisar a simplicidade no aprendizado e utilização da ferramenta, também é importante definir qual das ferramentas do Selenium está sendo considerada.

Como já informado anteriormente, o Selenium IDE é uma ferramenta de fácil instalação, pois funciona como um módulo de extensão do navegador web. Além disso, a ferramenta possui uma interface gráfica bem simples e intuitiva, permitindo que os usuários consigam utilizá-la com facilidade. Portanto, o aprendizado dessa ferramenta não deve representar uma grande dificuldade para os seus usuários.

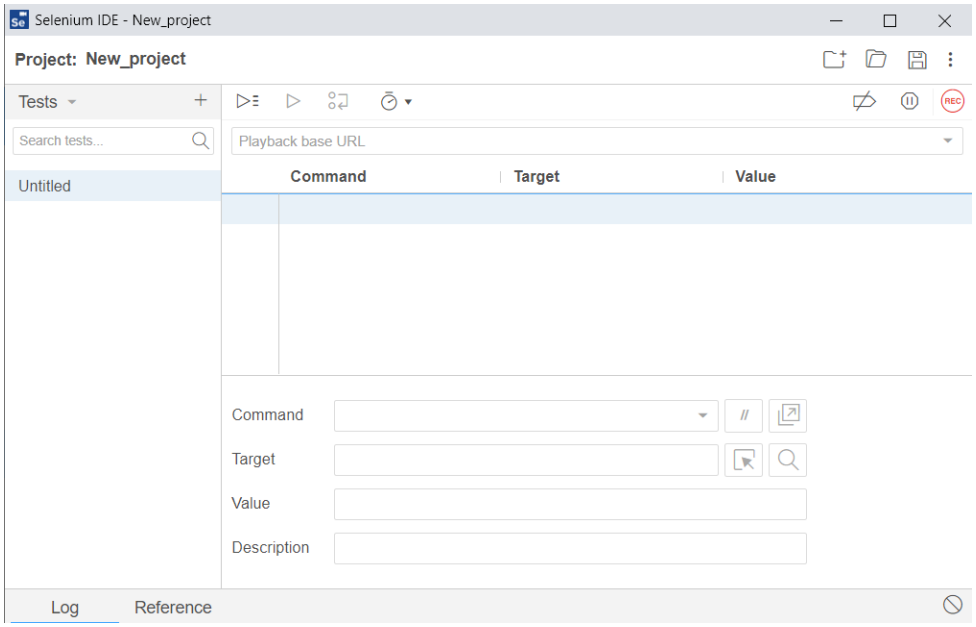

Figura 9.1: Interface gráfica do Selenium IDE.

O Selenium WebDriver possui alguns empecilhos que dificultam o processo de aprendizagem e utilização da ferramenta. Inicialmente, a própria instalação da ferramenta é mais complexa e depende que o usuário tenha alguns conhecimentos mais específicos de computação, como de ambientes integrados de desenvolvimento (IDE) e *drivers*, por exemplo. O seu uso também depende que o usuário já possua conhecimentos básicos de

programação, sendo portanto mais difícil. Existem diversos tutoriais para instalação e criação dos primeiros casos de teste para auxiliar os desenvolvedores que não possuam experiência com o Selenium WebDriver.

9.2.3 Avaliação do Selenium quanto à criação dos casos de teste

Essa seção busca responder a questão: "Quais são os recursos que a ferramenta possui para criação dos casos de teste?". Para esse aspecto, como o Selenium possui um conjunto de ferramentas de automação de testes, ele garante uma maior diversidade de recursos para a criação dos casos de teste.

O formato mais simples e chamativo, devido à facilidade que ele propõe na criação dos *scripts* de teste, é o módulo de captura do Selenium IDE. Ao usar essa funcionalidade, o usuário pode gravar pela ferramenta todas as suas interações na aplicação web, criando um *script* de teste que pode ser reexecutado posteriormente. Durante a gravação, as interações são armazenadas na forma de itens diferentes para cada comando, facilitando a legibilidade e manutenção do *script* de teste.

Em contrapartida, o Selenium WebDriver garante a criação dos *scripts* de teste por meio da codificação, ou seja, desenvolvimento do *script* com o uso de uma linguagem de programação. Apesar de mais complexo, esse recurso possibilita a criação de *scripts* de teste mais robustos, considerando a disponibilização de mais comandos, operações e integrações.

9.2.4 Avaliação do Selenium quanto à compreensão dos casos de teste

Essa seção busca responder a questão: "Os casos de teste criados por meio da ferramenta são de fácil compreensão por pessoas que não são profissionais da área?". Mais uma vez, é importante diferenciar as duas ferramentas do Selenium que foram verificadas no estudo para avaliar esse critério.

A utilização do Selenium IDE por profissionais que não são da área de tecnologia é possível, considerando que a sua interface gráfica é simples e permite que os testes sejam criados por meio de gravações. Para o Selenium WebDriver isso é mais difícil, já que os *scripts* são criados na forma de códigos, dificultando bastante a sua criação por pessoas que não são da área de computação ou que não possuam conhecimentos básicos de programação.

9.2.5 Avaliação do Selenium quanto ao mapeamento dos elementos da aplicação web

Essa seção busca responder a questão: "A ferramenta de automação utiliza identificadores ou posições baseadas em coordenadas para mapear os elementos da aplicação web?". Como o Selenium WebDriver é mais completo e possui mais recursos para criação de testes que o Selenium IDE, a análise dessa questão foi realizada com base nessa ferramenta.

De acordo com a documentação do Selenium WebDriver [12], existem diversos comandos e operações diferentes que podem ser usados para localizar os elementos da interface da aplicação web, também chamados de elementos web. Os identificadores, nomes de classe, nomes de atributos, texto do *link* e nomes de *tags* são alguns exemplos de objetos que podem ser utilizados para mapear os elementos. Para o projeto, os elementos foram mapeados corretamente por meio do nome de classe e por identificadores.

Portanto, o mapeamento dos elementos não é feito com base em coordenadas, o que representa uma grande vantagem. A utilização de coordenadas para mapear um objeto é frágil, pois qualquer alteração na disposição e exibição da interface pode gerar impactos nos casos de teste, o que não ocorre com a utilização de identificadores.

9.2.6 Avaliação do Selenium quanto à integração com banco de dados

Essa seção busca responder a questão: "A ferramenta de automação possui integração com banco de dados?". A proposta do Selenium é de realizar a automação de navegadores web para diversas finalidades diferentes, entre elas a realização de testes. Considerando que a sua estrutura se propõe a realizar a orientação dos navegadores, seu uso está mais relacionado com a interação da interface gráfica, não fazendo parte do seu escopo a interação com o banco de dados.

No entanto, considerando que o Selenium possui diversos recursos e integrações, ele pode ser combinado com *frameworks* e bibliotecas que possibilitam o acesso e realização de testes com o banco de dados. Muitas linguagens de programação que podem ser usadas na criação dos testes no Selenium, por exemplo, já possuem integração com sistemas gerenciadores de banco de dados.

No contexto deste projeto, nenhum recurso foi utilizado para integração com o banco de dados, considerando que esse não era o objetivo do trabalho.

9.2.7 Avaliação do Selenium quanto à integração com outros recursos e tecnologias

Essa seção busca responder a questão: "A ferramenta de automação pode ser integrada com outros recursos e tecnologias?". O Selenium possui uma comunidade de desenvolvimento bastante ativa disponibilizando diversos recursos que podem ser integrados ao Selenium WebDriver. Conforme informado anteriormente, ele já possui integração com diversos navegadores web para execução de testes, além de integração com diversas linguagens de programação distintas. Essa característica possibilita a extensão da ferramenta por meio do desenvolvimento de vários recursos diferentes para os *scripts* de teste.

A Figura 9.2 demonstra alguns componentes identficados na IDE Microsoft Visual Studio que podem ser incluídos no projeto de teste com o Selenium.

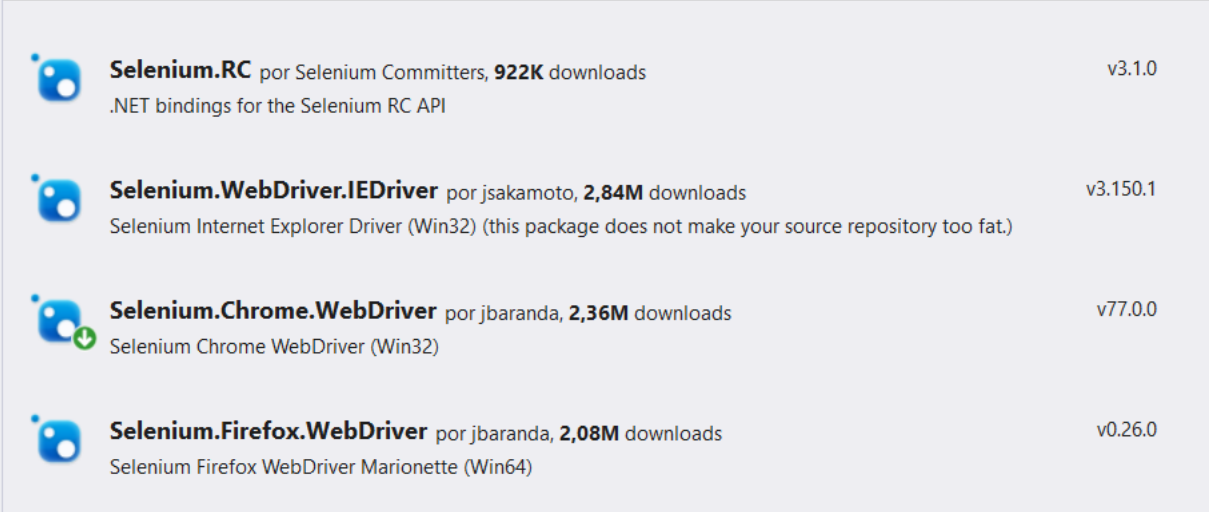

Figura 9.2: Exemplos de componentes do Selenium disponíveis para inclusão no projeto de teste.

Além disso, a própria documentação do Selenium [12] possui um tópico específico para abordar aspectos de extensão do Selenium por meio de estruturas criadas pelo usuário. Como os *scripts* são criados por meio da codificação em uma linguagem de programação, é possível desenvolver diferentes estruturas e complementos para integração com o Selenium WebDriver.

O Selenium IDE não possui muitos aspectos de integração, considerando a sua simplicidade no desenvolvimento dos testes.

Parte III

Considerações Finais

Capítulo 10

Conclusão

Este capítulo possui o intuito de apresentar, com base no referencial teórico e nos artefatos desenvolvidos ao longo do projeto, uma síntese dos resultados obtidos no trabalho. Para isso, ele foi dividido em três partes distintas. A primeira seção apresenta as conclusões observadas pelo estudo e as suas implicações. A segunda seção destaca algumas limitações identificadas ao longo do estudo. Por fim, a última seção apresenta sugestões para trabalhos futuros sobre o tema.

O principal objetivo do projeto era a criação de um arcabouço para avaliação da ferramenta de testes Selenium, baseado em alguns critérios estabelecidos previamente a partir da literatura. Entretanto, para que o processo de avaliação da ferramenta fosse realizado, inicialmente uma aplicação web foi desenvolvida utilizando o TDD, onde os *scripts* de teste foram desenvolvidos e executados com o Selenium.

10.1 Conclusões do estudo

Este trabalho buscou integrar diversos conceitos e práticas utilizados no processo de desenvolvimento de software. Nesse contexto, os elementos usados foram explicados e relacionados para deixar o projeto mais consistente e completo.

Inicialmente, o estudo realizado das aplicações web evidenciou o potencial desse tipo de sistema. A partir do referencial teórico, foi possível concluir que as aplicações web são sistemas utilizados por meio de um navegador web, podendo ser executadas localmente na máquina do usuário ou partindo de um servidor HTTP, onde seria necessário fazer uma requisição para uso do software. O crescimento da Internet e ampliação dos recursos disponíveis para as aplicações web propiciaram a consolidação desses sistemas, incentivando a criação de diversas ferramentas auxiliares e integrações para a web.

Com isso, buscou-se o planejamento e, em seguida, o desenvolvimento de uma aplicação web simples para o projeto. Considerando que a aplicação web não era o artefato principal

do trabalho, os *scripts* de teste também poderiam ser criados com base em outra aplicação web ou até mesmo a partir de alguma página da Internet sem grandes prejuízos para a utilização e validação do Selenium. No entanto, o desenvolvimento da aplicação web possibilitou uma melhor compreensão do seu funcionamento para a criação dos casos de teste, além de facilitar o desenvolvimento dos *scripts* de teste já que o código fonte e elementos estruturais da aplicação web eram conhecidos.

O desenvolvimento guiado por testes (TDD) foi a prática selecionada para o desenvolvimento da aplicação web, tendo em vista que essa prática possui grande foco em testes. Nesse contexto, essa abordagem possibilitou que o sistema fosse desenvolvido a partir da criação de testes simples, isolados e automáticos, instrumentando o desenvolvimento da aplicação web. Com o TDD, também foi possível definir as funcionalidades que seriam desenvolvidas por meio de histórias do usuário, assim como ocorre no método de desenvolvimento ágil XP. Essas características do TDD foram embasadas no referencial teórico do projeto.

Um dos requisitos para o uso do TDD é a automação dos testes, considerando que essa prática exige que os testes sejam executados frequentemente até a conclusão do processo de desenvolvimento. A automação de testes pode significar um aumento considerável na carga de trabalho, sendo necessário avaliar a sua necessidade para cada caso, pois ela nem sempre representa uma vantagem. Em aplicações com mudanças recorrentes na sua interface, por exemplo, não vale a pena realizar a automação dos testes, pois eles precisariam ser reescritos a cada mudança ocorrida. Para o contexto desse projeto, a automação foi necessária devido à prática de desenvolvimento escolhida.

Para realizar a automação dos testes da aplicação web, uma ferramenta de automação de testes foi utilizada para auxiliar nessa atividade. O Selenium foi escolhido para esse propósito, considerando que essa é uma ferramenta de código aberto para teste das aplicações web, com suporte para diversos navegadores e tecnologias.

Após o uso do Selenium para criação dos *scripts* de teste da aplicação web, as características da ferramenta foram avaliadas pelo desenvolvedor para identificar seus pontos positivos e negativos. Essa avaliação foi baseada em critérios específicos identificados na literatura sobre o tema, relacionados principalmente com os aspectos funcionais e de qualidade da ferramenta. Os critérios selecionados foram usados como inspiração para a criação de algumas questões, que foram respondidas de forma subjetiva para descrever as características e funcionalidades do Selenium.

A partir do resultado dessa avaliação, foi possível observar que o Selenium possui um conjunto de ferramentas diferentes e complementares, que fornecem uma grande variedade de recursos distintos para a automação de testes. A diversidade de recursos que esses componentes possuem possiblita que o Selenium seja usado tanto por profissionais mais

experientes que procuram realizar testes mais complexos, assim como por pessoas que não são da área de computação e que queiram automatizar testes mais simples. Sendo assim, é importante verificar o contexto do projeto de automação de testes para definir a ferramenta mais adequada a ser utilizada.

No cenário deste projeto, como a prática de desenvolvimento escolhida foi o TDD, a ferramenta utilizada na automação dos testes foi o Selenium WebDriver. Esse é um componente bastante poderoso do Selenium, possibilitando a criação dos *scripts* de teste por meio de programação. Além disso, ele também permite a integração com diversos recursos externos, bibliotecas e *frameworks*, seja para incluir customizações ao *script* de teste ou para integração com outras tecnologias, como sistemas gerenciadores de banco de dados, diferentes linguagens de programação ou navegadores web. Essa capacidade do Selenium WebDriver acaba tornando-o também mais complexo, desde o seu processo de instalação até a sua utilização, considerando que o desenvolvedor dos testes necessita ter conhecimentos de computação para codificação dos *scripts*. Portanto, a sua utilização acaba sendo mais difícil para profissionais que não são da área de computação.

Apesar do Selenium IDE não ser aplicável para automação de testes com o uso do TDD, também é importante destacar os recursos oferecidos por essa ferramenta. A utilização desse componente é bem mais simples, inclusive durante o seu processo de instalação, funcionando como uma extensão do navegador web. A sua interface gráfica permite a criação de testes por meio da captura das interações do usuário, possibilitando que qualquer pessoa que saiba utilizar a aplicação web consiga desenvolver os seus testes. Sendo assim, os testes criados por essa ferramenta acabam sendo mais simples, tendo em vista que o seu desenvolvimento é baseado na gravação durante a navegação no sistema. Entretanto, o Selenium IDE disponibiliza uma funcionalidade para exportação do *script* de teste criado, podendo ter o resultado com o Selenium WebDriver.

Desta forma, com base nas características listadas para as duas ferramentas do Selenium, é possível concluir que cada ferramenta se propõe a solucionar as desvantagens da outra ferramenta. Com isso, o Selenium IDE e o Selenium WebDriver acabam sendo ferramentas complementares, que podem ser utilizadas em conjunto para obter um resultado mais efetivo. O uso de somente uma das ferramentas representa vantagens em alguns aspectos e desvantagens em outros, conforme indicado anteriormente.

10.2 Limitações do estudo

Um dos principais fatores limitadores do estudo está relacionado à sua abrangência. Como o processo de planejamento e implementação da aplicação web foi realizado somente por uma pessoa, as suas características foram pautadas em uma visão mais restrita e limitada

pelo conhecimento e experiência do desenvolvedor. A participação de outras pessoas no processo de desenvolvimento, tanto dos testes como da aplicação, poderia enriquecer o estudo ao possibilitar análises por diferentes pontos de vista.

Essa restrição na abrangência do estudo também afetou o tamanho da aplicação desenvolvida e dos casos de teste criados para ela, considerando as limitações do desenvolvedor. A criação de mais *scripts* de teste poderia aperfeiçoar o processo de verificação da aplicação, tendo em vista que o projeto considerou apenas um teste funcional para cada funcionalidade desenvolvida.

Além disso, outro fator dificultador para o estudo foi o aspecto subjetivo associado ao processo de avaliação, que pode ser afetado por percepções individuais. Para minimizar esse problema, foram utilizados critérios de avaliação identificados na literatura sobre o tema, evitando também a definição de pesos e métricas específicos para os elementos de avaliação. Entretanto, como a avaliação é um processo de juízo de valor, ela está sujeita à interpretação do avaliador, representando uma limitação para o estudo.

Por fim, a avaliação da ferramenta foi feita com base em uma lista específica e limitada de critérios pré-definidos. A avaliação poderia ter sido diferente e mais completa com a análise de mais critérios de avaliação.

10.3 Sugestões de trabalhos futuros

Como sugestão para pesquisas futuras dentro desse tema, propõe-se a realização de estudos mais abrangentes, que possam incluir mais desenvolvedores para o projeto e mais critérios de avaliação a serem considerados, assim como o desenvolvimento de mais funcionalidades na aplicação web e mais casos de teste. Esse estudo também poderia considerar a escalabilidade da aplicação, possibilitando a avaliação de diferentes critérios, utilizando inclusive o Selenium Grid, outra ferramenta que compõe o conjunto de componentes do Selenium. A realização do estudo com mais pessoas poderia melhorar inclusive a avaliação da ferramenta, considerando que esse processo não estaria restrito ao juízo de valor de somente um avaliador.

Um estudo também pode ser realizado com a proposta de definir critérios e métricas padronizados para a avaliação de ferramentas de software, criando uma referência para o processo de avaliação de ferramentas de software. Nesse caso, a avaliação de software poderia ser baseada em parâmetros bem definidos, buscando evitar divergências na interpretação dos critérios de avaliação.

Referências

[1] San Murugesan and Yogesh Deshpande. *Web Engineering: Managing Diversity and Complexity of Web Application Development*. Springer, 2001. x, 9

[2] Athula Ginige and San Murugesan. Web engineering: a methodology for developing scalable, maintainable web applications. *Multimedia, IEEE*, 8:14 – 18, 2001. x, 8, 9, 10, 11, 12

[3] Ian Sommerville. *Engenharia de Software*. Pearson Addison-Wesley, São Paulo, 2007. x, 14, 15, 16, 17, 18, 19, 20, 21, 24

[4] Thaís Harumi Ussami. Incremental tests in a model based test driven development. *Dissertação (Mestrado) - Universidade Estadual de Campinas, Instituto de Computação*, 2016. x, 20, 24

[5] Lasse Koskela. *Test Driven: Practical Tdd and Acceptance Tdd for Java Developers*. Manning Publications Co., Greenwich, CT, USA, 2007. x, 32, 33, 54

[6] Marian Jureczko and Michał Młynarski. Automated acceptance testing tools for web applications using test-driven development. *Przegląd Elektrotechniczny*, 86:198–202, 2010. x, 34, 42, 44, 45, 59

[7] Anne Mette Jonassen Hass. *Guide to Advanced Software Testing*. Artech House, Inc., Norwood, MA, USA, 2008. xii, 1, 30, 31, 32

[8] Macario Polo, Pedro Reales Mateo, Mario Piattini, and Christof Ebert. Test automation. *Software, IEEE*, 30:84–89, 2013. xii, 30, 31, 32

[9] Rigzin Angmo and Monika Sharma. Selenium tool: A web based automation testing framework. In *International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)*, 2014. xii, 47, 48, 49

[10] Mary Jean Harrold. Testing: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 61 – 72, New York, NY, USA, 2000. ACM. 1

[11] Lars-Ola Damm, Lars Lundberg, and David Olsson. Introducing test automation and test-driven development: An experience report. *Electronic Notes in Theoretical Computer Science*, 116:3–15, 2005. 1, 33

[12] SeleniumHQ. Selenium documentation. Disponível em: `https://docs.seleniumhq.org/docs/`. Acesso em: 23 de outubro de 2019. 2, 46, 47, 48, 72, 76, 77

[13] Gustavo Rossi, Oscar Pastor, Daniel Schwabe, and Luis Olsina. *Web Engineering: Modelling and Implementing Web Applications*. Human–Computer Interaction Series. Springer London, 2007. 6, 7, 8, 12

[14] Mehdi Jazayeri. Some trends in web application development. In *2007 Future of Software Engineering*, pages 199 – 213, Washington, DC, USA, 2007. IEEE Computer Society. 6, 7

[15] Filippo Ricca and Paolo Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 25– 34, 2001. 11

[16] SWEBOK. Guide to the software engineering body of knowledge (swebok v3). Disponível em: `http://www.computer.org/portal/web/swebok/`, 2004. Acesso em: 15 de maio de 2019. 11, 13, 16, 17, 28

[17] SEVOCAB. Software and systems engineering vocabulary. Disponível em: `https://pascal.computer.org/`. Acesso em: 15 de maio de 2019. 13, 15, 26

[18] Roger Pressman. *Software Engineering: a practitioner's approach*. McGraw-Hill, 7 edition, 2009. 14, 15, 16, 17, 25

[19] Mary-Luz Sánchez Gordón and Rory V. O'Connor. Understanding the gap between software process practices and actual practice in very small companies. *Software Quality Journal*, 24:549–570, 2016. 14

[20] David Janzen and Hossein Saiedian. Test-driven development concepts, taxonomy, and future direction. *Computer*, 38:43–50, 2005. 15, 19, 20

[21] Kent Beck. Aim, fire. *IEEE Software*, 18:87–89, 2001. 15, 19

[22] MANIFESTO. Manifesto para o desenvolvimento ágil de software. Disponível em: `https://www.manifestoagil.com.br/`, 2001. Acesso em: 10 de junho de 2019. 16

[23] Kent Beck. *Programação extrema (XP) explicada: acolha as mudanças*. Bookman, Porto Alegre, 2004. 18, 19

[24] Mauricio Finavaro Aniche, Thiago Miranda Ferreira, and Marco Aurélio Gerosa. What concerns beginner test-driven development practitioners: a qualitative analysis of opinions in an agile conference. In *2nd Brazilian Workshop on Agile Methods*, 2011. 19, 22

[25] Mauricio Finavaro Aniche. Como a prática de tdd influencia o projeto de classes em sistemas orientados a objetos. *Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo*, 2012. 19, 20, 21

[26] Burak Turhan, Lucas Layman, Madeline Diep, Forrest Shull, and Hakan Erdogmus. How effective is test driven development. In *Making Software*. O'Reilly Media, Inc., 2010. 19, 21, 23, 53

[27] Zeba Khanam and Mohammed Najeeb Ahsan. Evaluating the effectiveness of test driven development: advantages and pitfalls. *International Journal of Applied Engineering Research*, 12(18):7705–7716, 2017. 20, 22, 24

[28] Ayse Tosun, Muzamil Ahmed, Burak Turhan, and Natalia Juristo. On the effectiveness of unit tests in test-driven development. In *Proceedings of the 2018 International Conference on Software and System Process*, pages 113–122, New York, NY, USA, 2018. ACM. 21, 24

[29] Joelma Choma, Eduardo Guerra, and Tiago Da Silva. Developers' initial perceptions on tdd practice: a thematic analysis with distinct domains and languages. In *Agile Processes in Software Engineering and Extreme Programming*, pages 68 – 85. Springer International Publishing, 2018. 23

[30] Mauricio Finavaro Aniche and Marco Aurélio Gerosa. Most common mistakes in test-driven development practice: results from an online survey with developers. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 469–478, 2010. 23

[31] Raja Parasuraman, Thomas Sheridan, and Christopher Wickens. A model for types and levels of human interaction with automation. *IEEE transactions on systems, man, and cybernetics. Part A, Systems and humans : a publication of the IEEE Systems, Man, and Cybernetics Society*, 30:286–97, 2000. 26, 27

[32] Raja Parasuraman and Victor Riley. Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, 39:230–253, 1997. 26, 27

[33] Britannica Academic. Automation. https://academic-eb-britannica.ez54.periodicos.capes.gov.br/levels/collegiate/article/automation/109393. 27

[34] Thomas Sheridan and Raja Parasuraman. Human-automation interaction. *Reviews of Human Factors and Ergonomics*, 1:89–129, 2005. 27

[35] Elfriede Dustin, Jeff Rashka, and John Paul. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 28, 29

[36] Katja Karhu, Tiina Repo, Ossi Taipale, and Kari Smolander. Empirical observations on software testing automation. In *2009 International Conference on Software Testing Verification and Validation*, pages 201 – 209, 2009. 28, 29

[37] Ana Paula Cavalcanti, Silvio Meira, and Marcos Gomes. Towards a maturity model in software testing automation. In *The Ninth International Conference on Software Engineering Advances*, 2014. 29, 30

[38] Diego Clerissi, Maurizio Leotta, Gianna Reggio, and Filippo Ricca. Test driven development of web applications: A lightweight approach. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 25–34, 2016. 33

[39] Michael Scriven. The logic of evaluation. In *Ontario Society for the Study of Argumentation*. Scholarship at Uwindsor, 2007. 35

[40] Peter Baumgartner and Sabine Payr. Methods and practice of software evaluation. the case of the european academic software award. pages 44–50. ED-MEDIA 97 - World Conference on Educational Multimedia and Hypermedia, 1997. 36

[41] Catherine Fritz and Bradley Carter. A classification and summary of software evaluation and selection methodologies. Technical report, Mississippi State, MS, USA, 1994. 36

[42] Anil Jadhav and Rajendra Sonar. Evaluating and selecting software packages: A review. *Information and Software Technology*, 51(3):555 – 563, 2009. 36, 37, 41

[43] Harsh Bajaj. Choosing the right automation tool and framework is critical to project success. *Infosys Limited*, 2015. 41

[44] Tibor Illes, Agnès Herrmann, Barbara Paech, and Julius Rückert. Criteria for software testing tool evaluation: a task oriented view. In *Proceedings of the 3rd World Congress for Software Quality*, volume 2, pages 213–222, 2005. 42